AODX Document Exchange System

Administration & User's Guide Version 1.0.0

Microsoft Dynamics 365 for Finance and Supply Chain Management

Table of Contents

Copyright Notice

Atlantic Oak LLC.
5 Concourse Parkway
Suite 3000
Atlanta, GA 30328
United States of America
Phone: +1 (470) 582-0912
E-Mail: support@atlantic-oak.com
Website: https://atlantic-oak.com

Version History

Version	Date	Description
001.00.00	06/02/2021	First release.

Introduction

The AODX Document Exchange System is a Microsoft Dynamics 365 for Finance and Supply Chain Management module that allows D365 to convert any of its database tables into XML or CSV files, or from CSV and XML files into tables, and then import or export these CSV and XML files directly from and to SFTP, FTP, FTPS, Azure Storage and Azure Files servers.

The AODX system can be used to integrate Microsoft Dynamics 365 for Finance and Supply Chain Management with external applications that can read and write to a file system.

The AODX system can import and export these types of files:

- *Coma Separated Values or CSV files:* A type of text file that separates fields with a coma or any other character such as a tab and separates rows with a carriage return and a line feed.
- *Extensible Markup Language or XML files:* A type of text file that describes the structure of data using tags. It is much more powerful than a CSV file and a lot more information can be packed into a single file compared to a CSV file.

The AODX system can write files to and read files from these file servers:

- *SSH File Transfer Protocol or SFTP:* A network protocol that provides secure file access and transfer through a stream. It is an extension of the Secure Shell Protocol (SSH) version 2.0.
- *File Transfer Protocol or FTP:* It is an old file transfer protocol that was developed in the 1970's and ran on NCP before it was replaced by TCP/IP (the internet protocol in use today) in 1985. It provides unsecured transfer of files between a client and a server. Passwords, usernames and files are not secure and can be intercepted.
- *File Transfer Protocol Secure or FTPS:* An extension to the FTP protocol that adds support for the Transfer Layer Security cryptographic protocol. This protocol addresses security concerns on the FTP protocol.
- *Azure BLOB Storage:* Is Microsoft's file storage in the cloud. It can store massive amounts of files of every size in different formats.
- *Azure Files:* Is an extension to Azure Storage that enables you to set up network file shares that can be accessed by using the standard Server Message Block (SMB) protocol.

In the AODX system users can define the way in which to read or write a file without writing code. Users can change those instructions if needed without having to bring down the system. If necessary, CSV and XML file import/export can also be hard coded in X++ via extensions to the AODX system model.

Trial Version

Visit https://atlantic-oak.com/TrialVersions to request your trial version. You can test trial the AODX system for a period of one month and you can also request additional extensions to this trial period if needed.

Prospective clients are entitled to the same level of support as our regular customers.

Support

If you require assistance, experience problems or if you have any questions or comments send us an e-mail to support@atlantic-oak.com.

Our support services include answering questions about the existing functionality of our applications, small code snippets, general suggestions and expedited bug fixing. Regular support requires competency in the technology and development environment being used (Dynamics 365 for Finance and Operations, Visual Studio and X++).

Installation Instructions

You will receive the AODX System and updates to it via deployment packages. A deployment package is a zip file. To install a package on a development environment, follow these instructions:

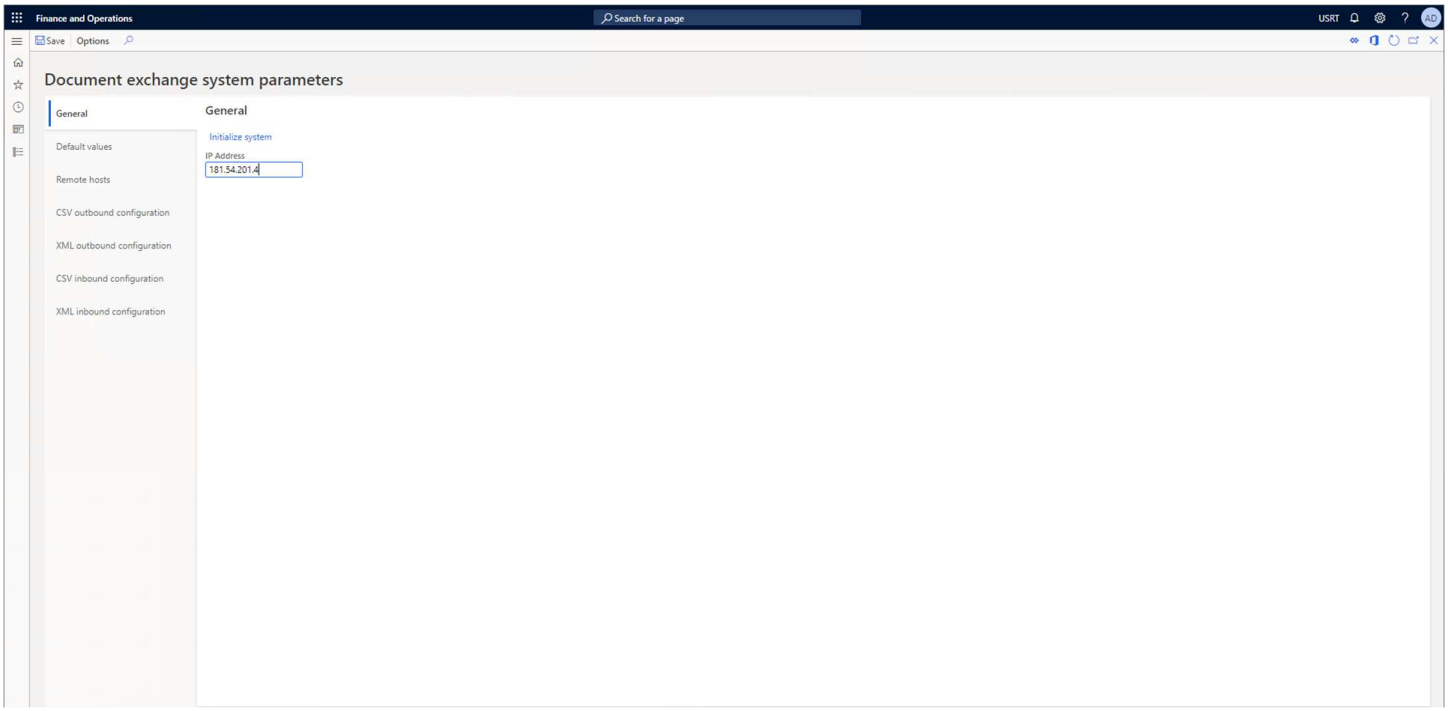
https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/deployment/install-deployable-package

Basic Setup

Follow this procedure to setup the AODX system in every legal entity in the system.

Initialize system tables.

Go to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters*



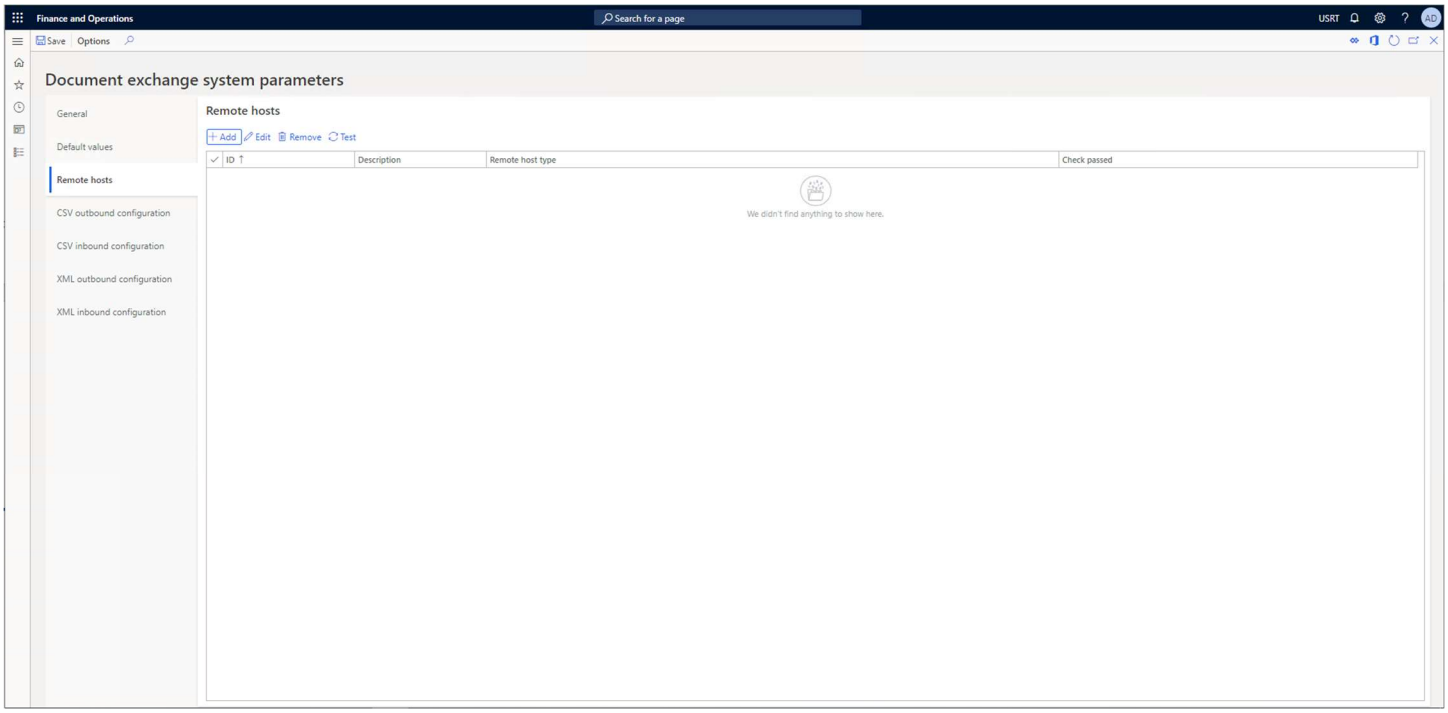
In the *General* tab click on the *Initialize system* button. This procedure will update some AODX system tables with new information. It is important to run this procedure whenever the underlying database structure of the Dynamics 365 system has changed, i.e.: new tables have been created or existing ones have been deleted.

Create an entry in Remote Hosts for the temporary Azure Storage account.

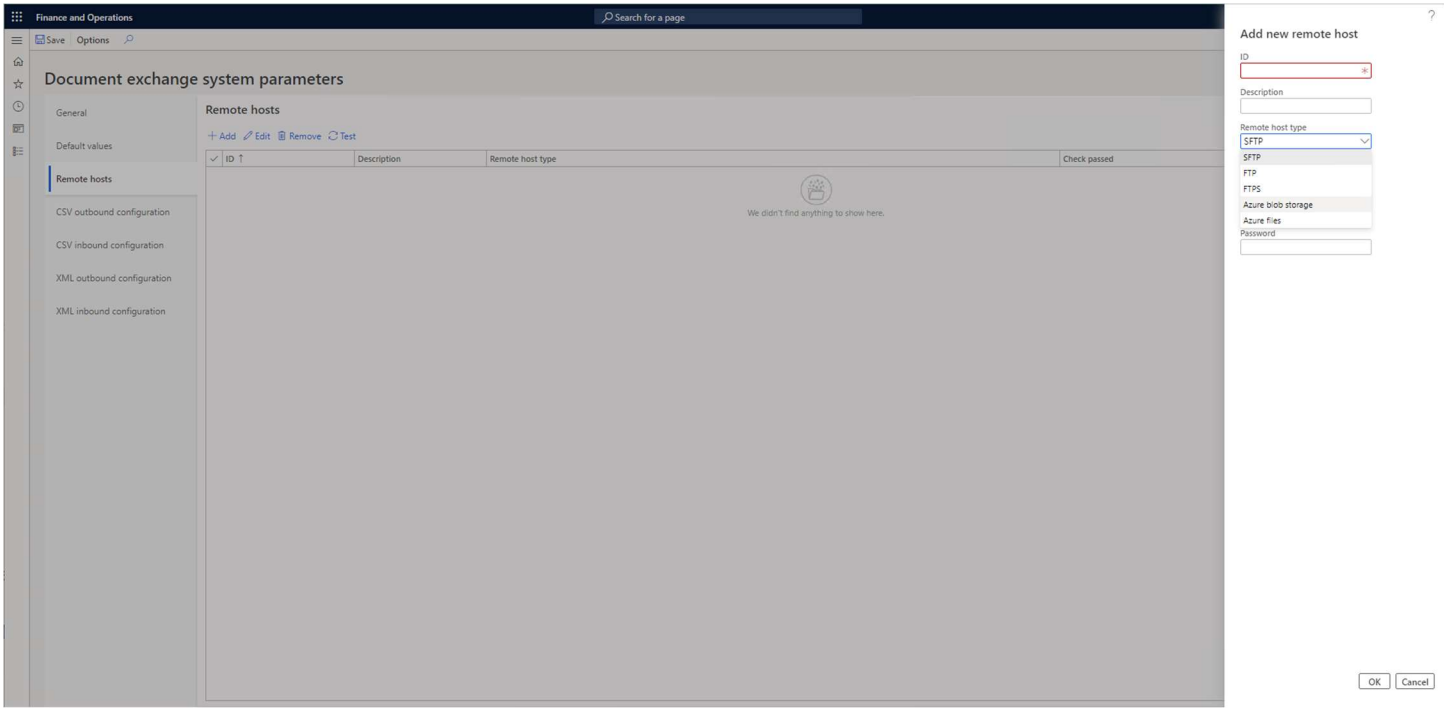
The AODX system requires an Azure Storage account to store incoming and outgoing files for reference and auditing purposes. Before you continue, you must create an Azure Storage account or designate an existing one for this purpose. You can find more information on how to create an Azure Storage account by visiting this link:

https://docs.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal

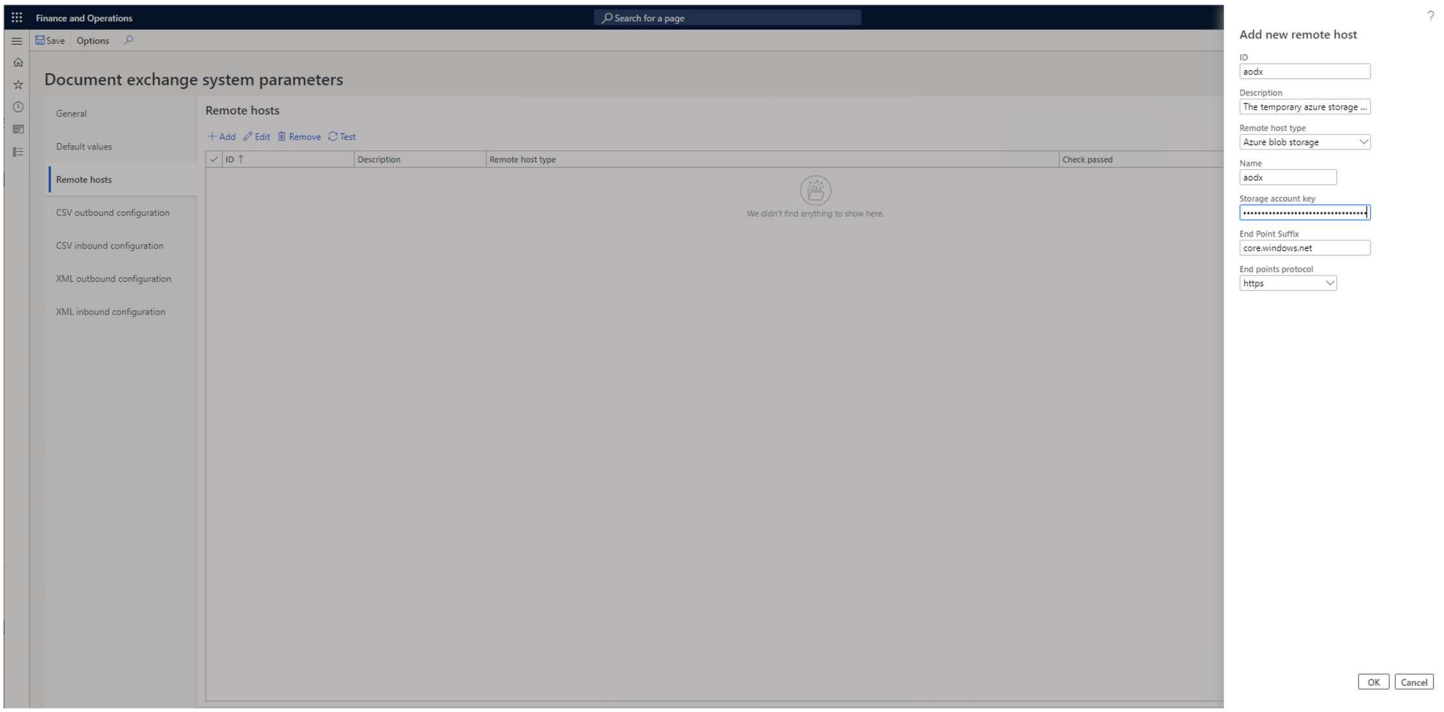
Once you have your Azure Account setup and ready go to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and click on the *Remote hosts* tab.



Click on the *Add* button.



From the *Remote host type* dropdown select *Azure blob storage*.



Fill in these fields:

ID: Any alphanumeric identifier that will uniquely identify this remote host (required). In this case the *ID* matches the *Name* but it doesn't have to.

Description: A meaningful description of the remote host (optional).

Name: The name of the Azure Storage account (required).

Storage account key: The storage account key is a credential that together with the account name prevents unauthorized access to the account (required).

End point suffix: The end point suffix that determines the Azure cloud region, it is generally core.windows.net or core.chinacloudapi.cn (required).

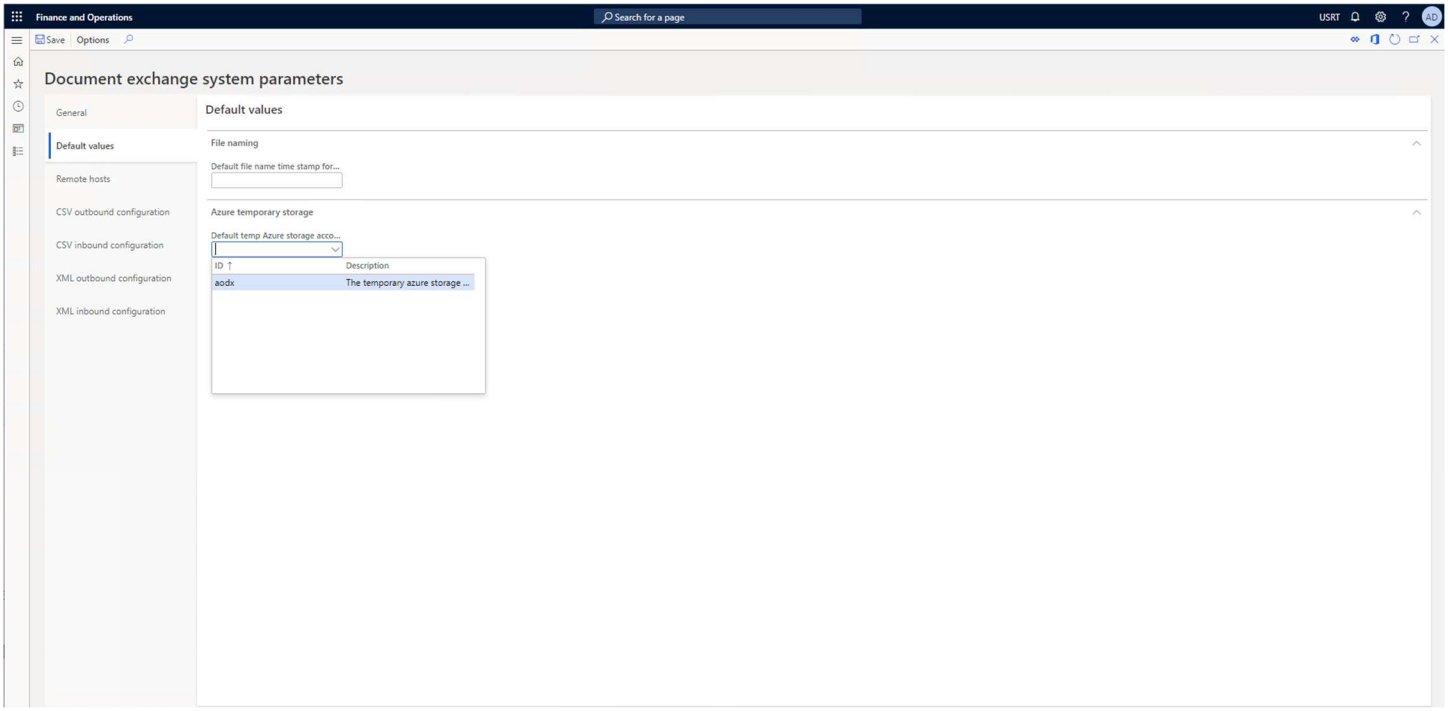
End points protocol: The protocol that the storage account uses to communicate (required). http (unsecure) or https (secure and encrypted).

Once you have completed filling out the required fields click on the *OK* button.

Set the default temporary Azure Storage account.

Since the AODX system always requires a temporary Azure Storage account, if none is found on the document version or the document, the system will use the default that is specified in the *Document Exchange System Parameters* form.

To set the temporary default Azure Storage account go to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and click on the *Default values* tab.



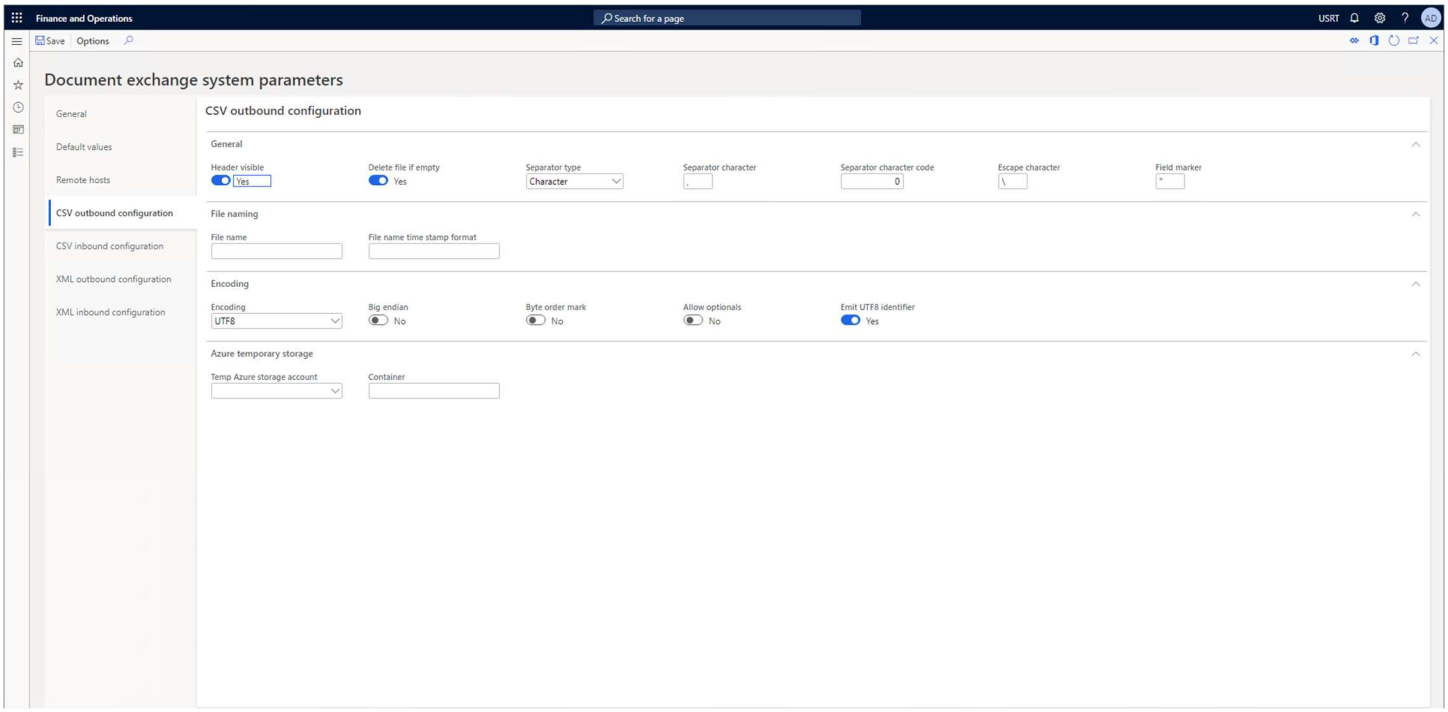
Select the storage account that you have created in the previous step from the dropdown.

CSV Outbound Configuration Setup

The AODX system has four document types, one of them is CSV outbound files.

The system default values that are applied to an outbound csv document version can be configured by going to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and clicking on the *CSV outbound configuration* tab.

The settings entered on this screen will be overridden by the CSV outbound configuration settings specified on a Document which in turn will be overridden in the CSV outbound configuration settings specified on a Document version.



General.

Header visible: Yes or No. If yes, the field headers are going to be displayed on the first line of the output csv file.

Delete file if empty: Yes or No. If yes, the system will not create a file if it has no records.

Separator type: Character or Character code. If set to Character the you can specify the character to use in the Separator character field. If set to Character code you can specify an ASCII character code in the Separator character code field.

Separator character: Any valid separator character that separates the fields. Do not use characters that might be used inside any of the fields.

Separator character code: Any valid ASCII character to be used as the field separator character.

Escape character: Character that denotes an escape sequence.

Field marker: Character that marks the beginning and end of a field.

File naming.

File name: The naming structure of the outgoing csv file. An asterisk(*) will be replaced with the data and time defined in the File name time stamp format.

File name time stamp format: The structure of the date/time stamp that will be applied to the outgoing file. Follows the formatting for .NET Custom date and time format strings.

Encoding.

Encoding: ASCII, BigEndianUnicode, Default, UTF32, UTF7, UTF8 or Unicode. The text encoding option.

Big endian: Yes or No. If set to Yes, the most significant byte (the "big end") of the data is placed at the byte with the lowest address. Only used when UTF32 or Unicode are selected in the Encoding field.

Byte order mark: Yes or No. If set to Yes, places the special Unicode character, U+FEFF (Byte order mark) at the start of the text stream. Only used when UTF32 or Unicode are selected in the Encoding field.

Allow optionals: Yes or No. If set to Yes, allows optional characters for UTF7. Only used when UTF7 is selected in the Encoding field.

Emit UTF8 identifier: Yes or No. If set to Yes, places the special Unicode character, U+FEFF (Byte order mark) at the start of the text stream. Only used when UTF8 is selected in the Encoding field.

Azure temporary storage.

Temp Azure storage account: The ID of the remote host that will be used as the temporary Azure Storage account for all outgoing CSV files. It overrides the setting on the Document exchange system parameters tab and is overridden by any setting on the Document or the Document version.

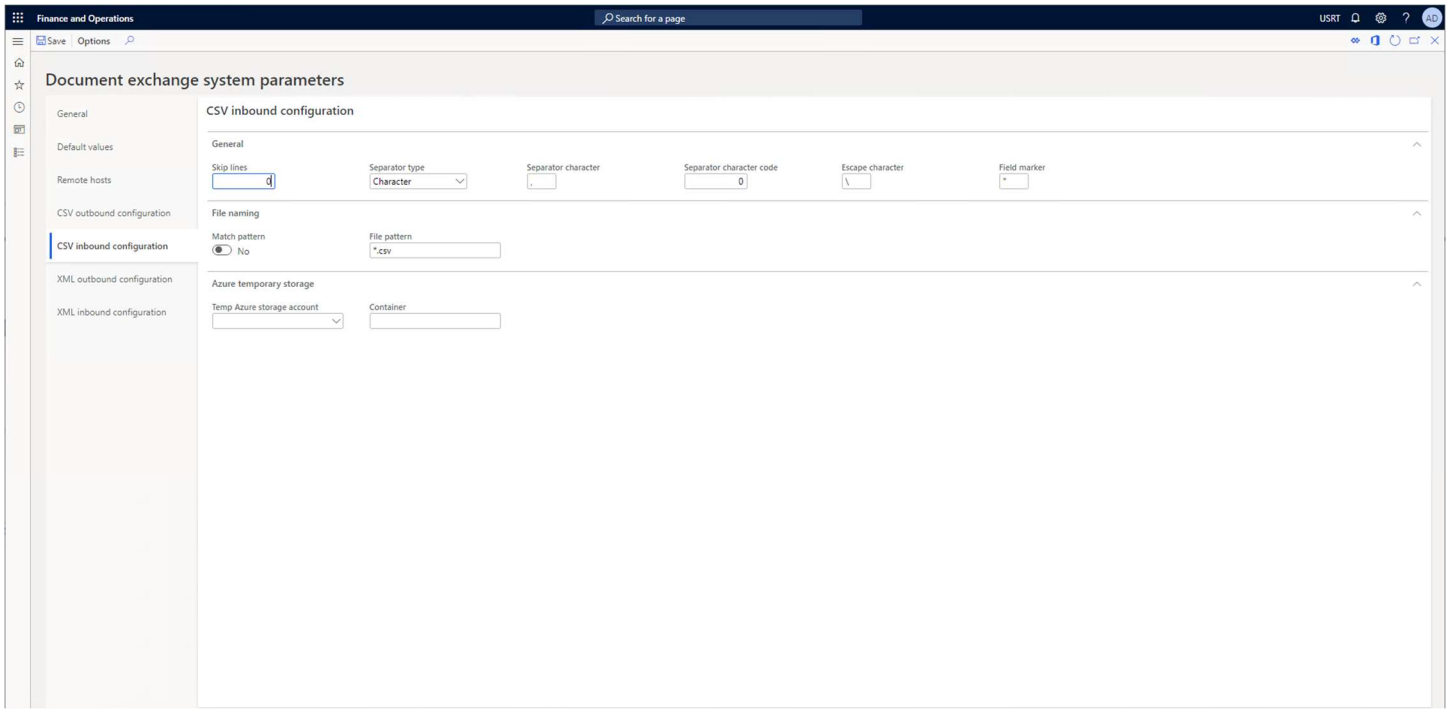
Container: The name of the container for all outgoing CSV files. Is overridden by any setting on the Document or the Document version.

CSV Inbound Configuration Setup

The AODX system has four document types, one of them is CSV inbound files.

The system default values that are applied to an inbound CSV document version can be configured by going to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and clicking on the *CSV inbound configuration* tab.

The settings entered on this screen will be overridden by the CSV inbound configuration settings specified on a Document which in turn will be overridden in the CSV inbound configuration settings specified on a Document version.



General.

Skip lines: Skip n number of lines at the beginning of the incoming CSV file.

Separator type: Character or Character code. If set to Character the you can specify the character to use in the Separator character field. If set to Character code you can specify an ASCII character code in the Separator character code field.

Separator character: The character that separates the fields.

Separator character code: Any valid ASCII character that is used as the field separator character.

Escape character: Character that denotes an escape sequence.

Field marker: Character that marks the beginning and end of a field.

File naming.

Match pattern: Yes or No. If set to Yes, the system will only pick up files that match the pattern in field File pattern.

File pattern: The file pattern to match files against. An asterisk(*) stands for a wildcard or any number of alphanumeric characters.

Azure temporary storage.

Temp Azure storage account: The ID of the remote host that will be used as the temporary Azure Storage account for all incoming CSV files. It overrides the setting on the Document exchange system parameters tab and is overridden by any setting on the Document or the Document version.

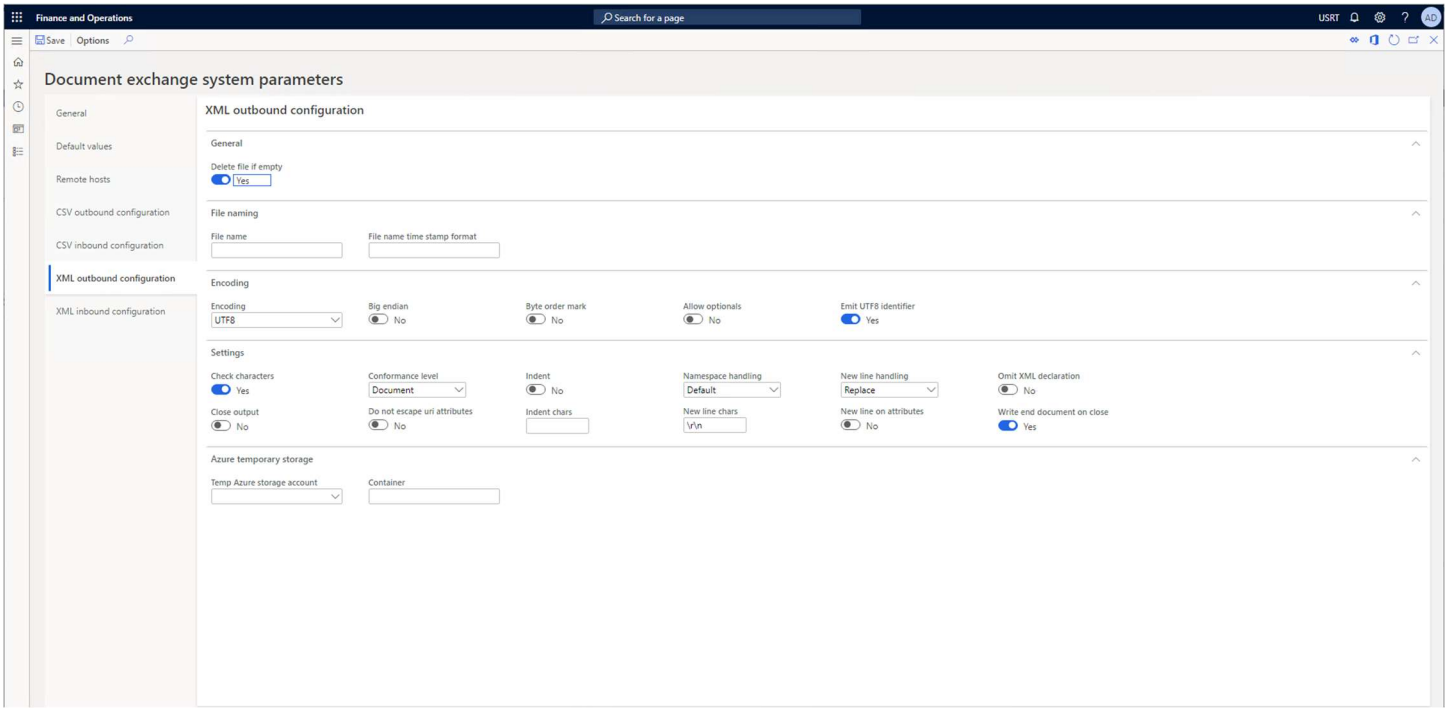
Container: The name of the container for all incoming CSV files. Is overridden by any setting on the Document or the Document version.

XML Outbound Configuration Setup

The AODX system has four document types, one of them is XML outbound files.

The system default values that are applied to an outbound XML document version can be configured by going to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and clicking on the *XML outbound configuration* tab.

The settings entered on this screen will be overridden by the XML outbound configuration settings specified on a Document which in turn will be overridden in the XML outbound configuration settings specified on a Document version.



General.

Delete file if empty: Yes or No. If yes, the system will not create a file if it has no records.

File naming.

File name: The naming structure of the outgoing XML file. An asterisk(*) will be replaced with the data and time defined in the File name time stamp format.

File name time stamp format: The structure of the date/time stamp that will be applied to the outgoing file. Follows the formatting for .NET Custom date and time format strings.

Encoding.

Encoding: ASCII, BigEndianUnicode, Default, UTF32, UTF7, UTF8 or Unicode. The text encoding option.

Big endian: Yes or No. If set to Yes, the most significant byte (the "big end") of the data is placed at the byte with the lowest address. Only used when UTF32 or Unicode are selected in the Encoding field.

Byte order mark: Yes or No. If set to Yes, places the special Unicode character, U+FEFF (Byte order mark) at the start of the text stream. Only used when UTF32 or Unicode are selected in the Encoding field.

Allow optionals: Yes or No. If set to Yes, allows optional characters for UTF7. Only used when UTF7 is selected in the Encoding field.

Emit UTF8 identifier: Yes or No. If set to Yes, places the special Unicode character, U+FEFF (Byte order mark) at the start of the text stream. Only used when UTF8 is selected in the Encoding field.

Settings.

Check characters: Yes or No. If set to Yes, a check will be performed making sure that characters are in the legal XML character set, as defined by W3C.

Close output: Yes or No. If set to Yes, the underlying stream will be closed when the XML writer is closed.

Conformance level: Auto, Fragment or Document. Sets the level of conformance for the XML writer.

Do not escape uri attributes: Yes or No. If set to Yes, the XML writer will not escape uri attributes.

Indent: Yes or No. If set to Yes, indents output for easier readability.

Indent chars: The character string to use when indenting.

Namespace handling: Default or Omit duplicates. If set to Omit duplicates the XML writer will remove duplicate namespace declarations.

New line chars: The character string to use for line breaks.

New line handling: Replace, Entitize or None.

New line on attributes: Yes or No. If set to Yes, will write attributes on a new line.

Omit XML declaration: Yes or No. If set to Yes, the XML declaration will be omitted.

Write end document on close: Yes or No. If set to Yes, when the close method is called, the XML writer will close any open tags.

Azure temporary storage.

Temp Azure storage account: The ID of the remote host that will be used as the temporary Azure Storage account for all outgoing XML files. It overrides the setting on the Document exchange system parameters tab and is overridden by any setting on the Document or the Document version.

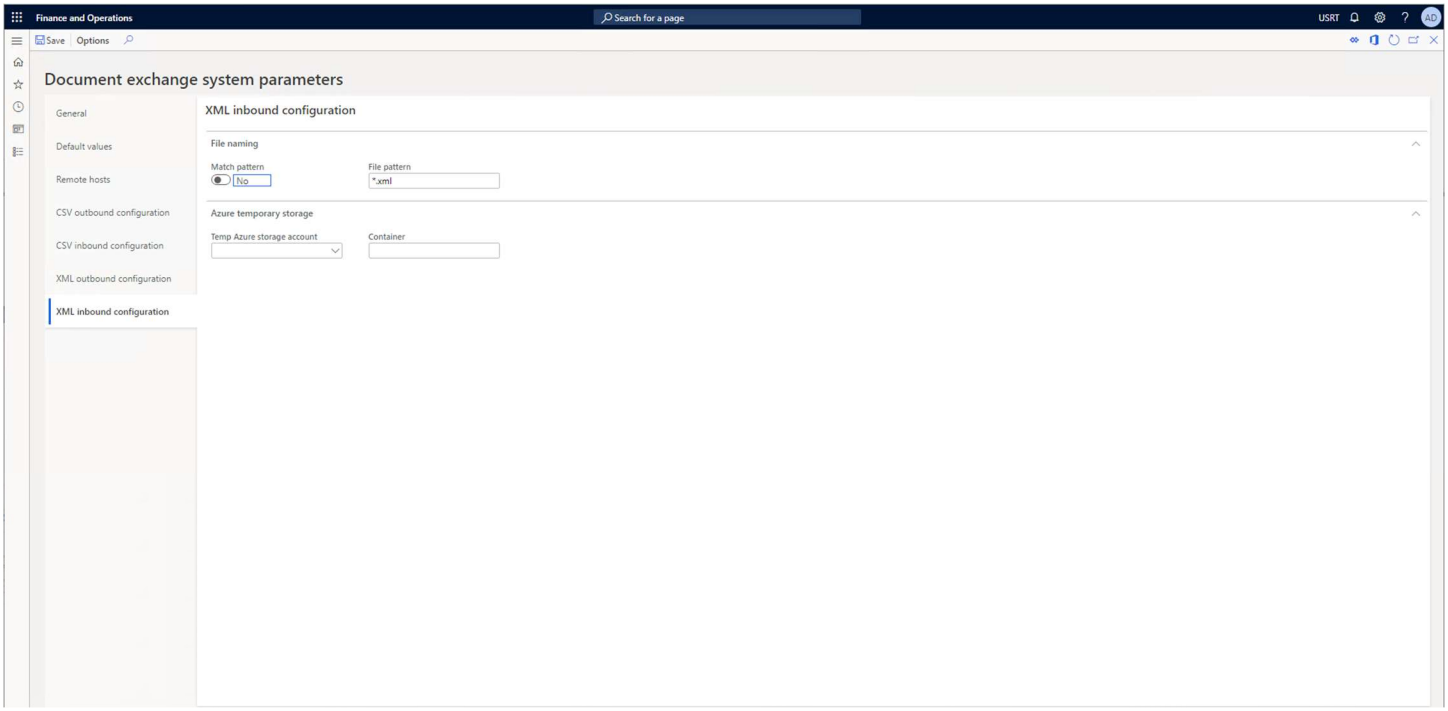
Container: The name of the container for all incoming XML files. Is overridden by any setting on the Document or the Document version.

XML Inbound Configuration Setup

The AODX system has four document types, one of them is XML inbound files.

The system default values that are applied to an inbound XML document version can be configured by going to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and clicking on the *XML inbound configuration* tab.

The settings entered on this screen will be overridden by the XML inbound configuration settings specified on a Document which in turn will be overridden in the XML inbound configuration settings specified on a Document version.



File naming.

Match pattern: Yes or No. If set to Yes, the system will only pick up files that match the pattern in field File pattern.

File pattern: The file pattern to match files against. An asterisk(*) stands for a wildcard or any number of alphanumeric characters.

Azure temporary storage.

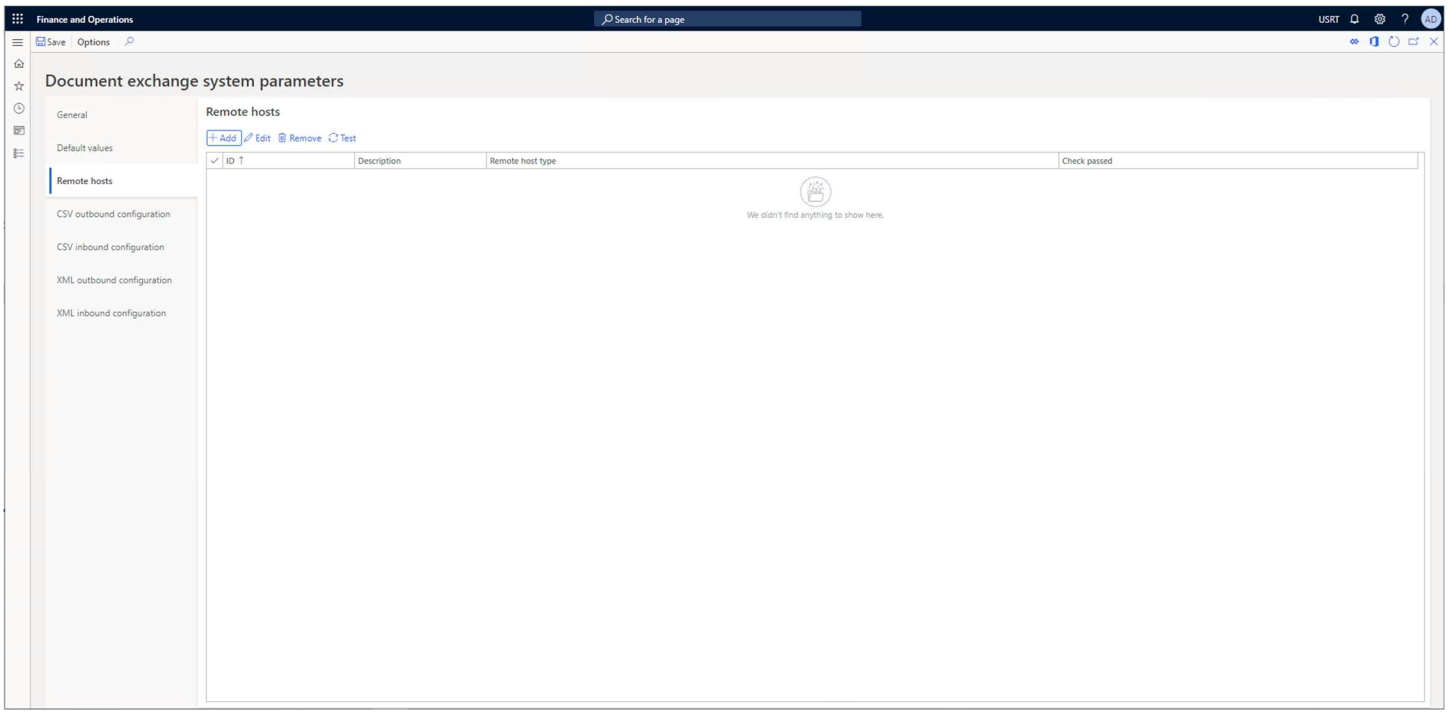
Temp Azure storage account: The ID of the remote host that will be used as the temporary Azure Storage account for all incoming CSV files. It overrides the setting on the Document exchange system parameters tab and is overridden by any setting on the Document or the Document version.

Container: The name of the container for all incoming CSV files. Is overridden by any setting on the Document or the Document version.

Remote Hosts Setup

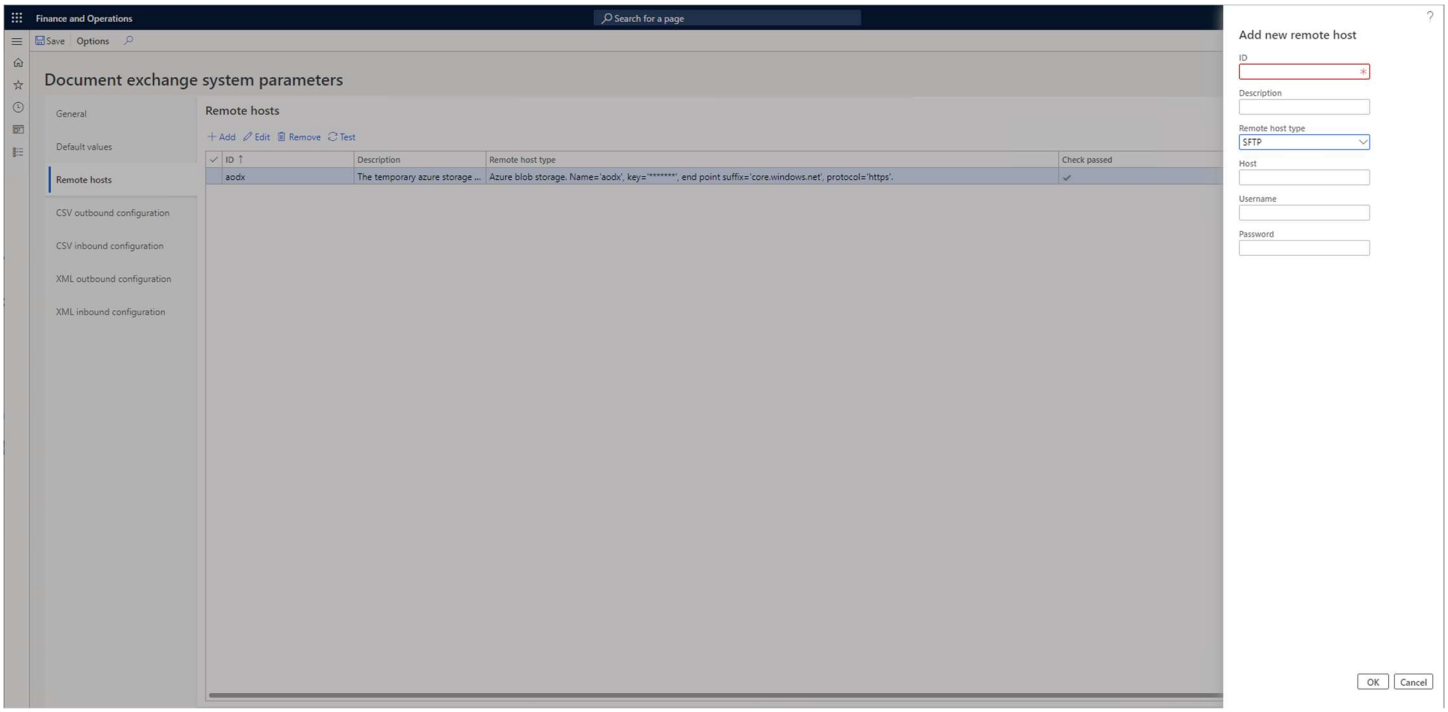
The AODX system supports importing and exporting to five different remote host types. To carry out these import and export operations the remote hosts must be configured in the *Document Exchange System Parameters* form. Each type of remote host has a different way of being configured.

To add a remote host, go to: *Modules -> Document Exchange System -> Setup -> Document Exchange System Parameters* and click on the *Remote hosts* tab.



Click on the *Add* button.

Add an SFTP server



Fill in these fields:

ID: Any alphanumeric identifier that will uniquely identify this remote host (required).

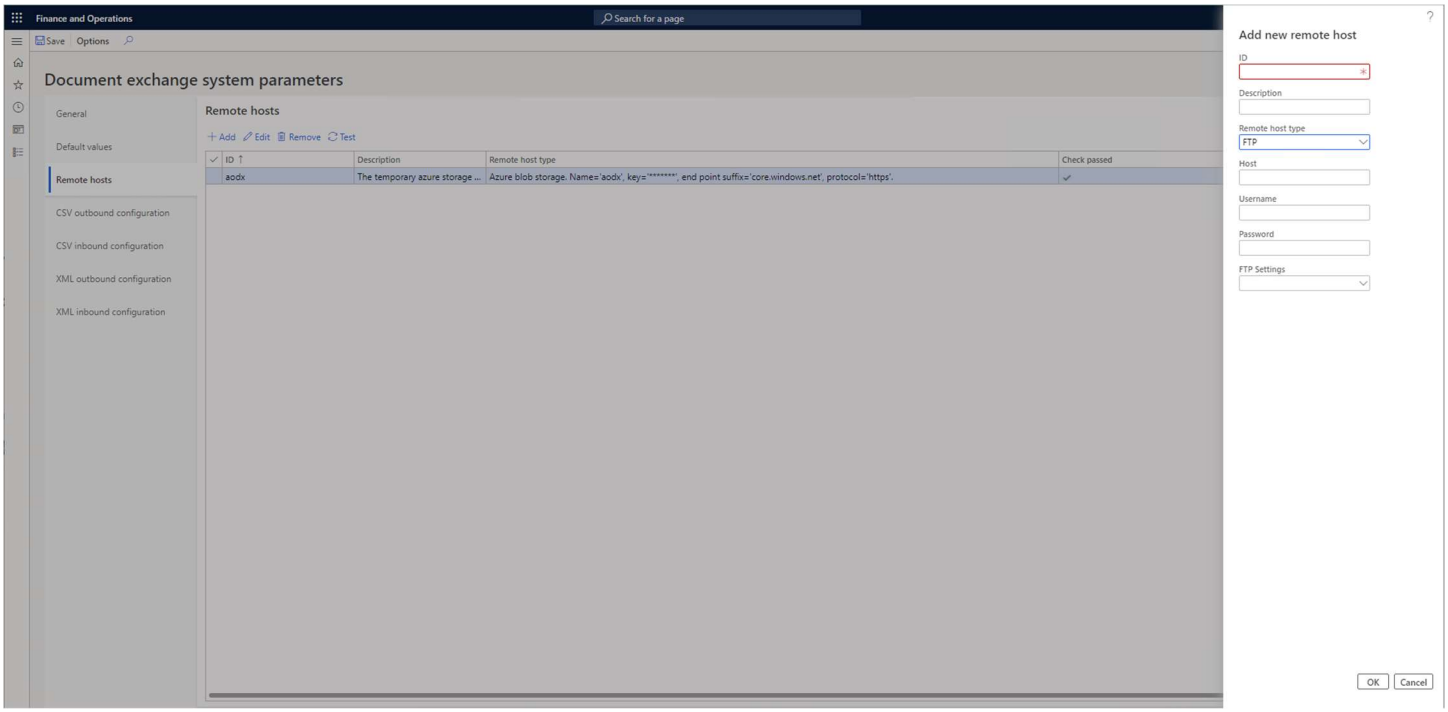
Description: A meaningful description of the remote host (optional).

Host: The IP address or DNS name of the SFTP server (example: 56.7.8.9 or server.somedomain.com - required).

Username: The username part of the credentials (required).

Password: The password part of the credentials (required).

Add an FTP or FTPS server



Fill in these fields:

ID: Any alphanumeric identifier that will uniquely identify this remote host (required).

Description: A meaningful description of the remote host (optional).

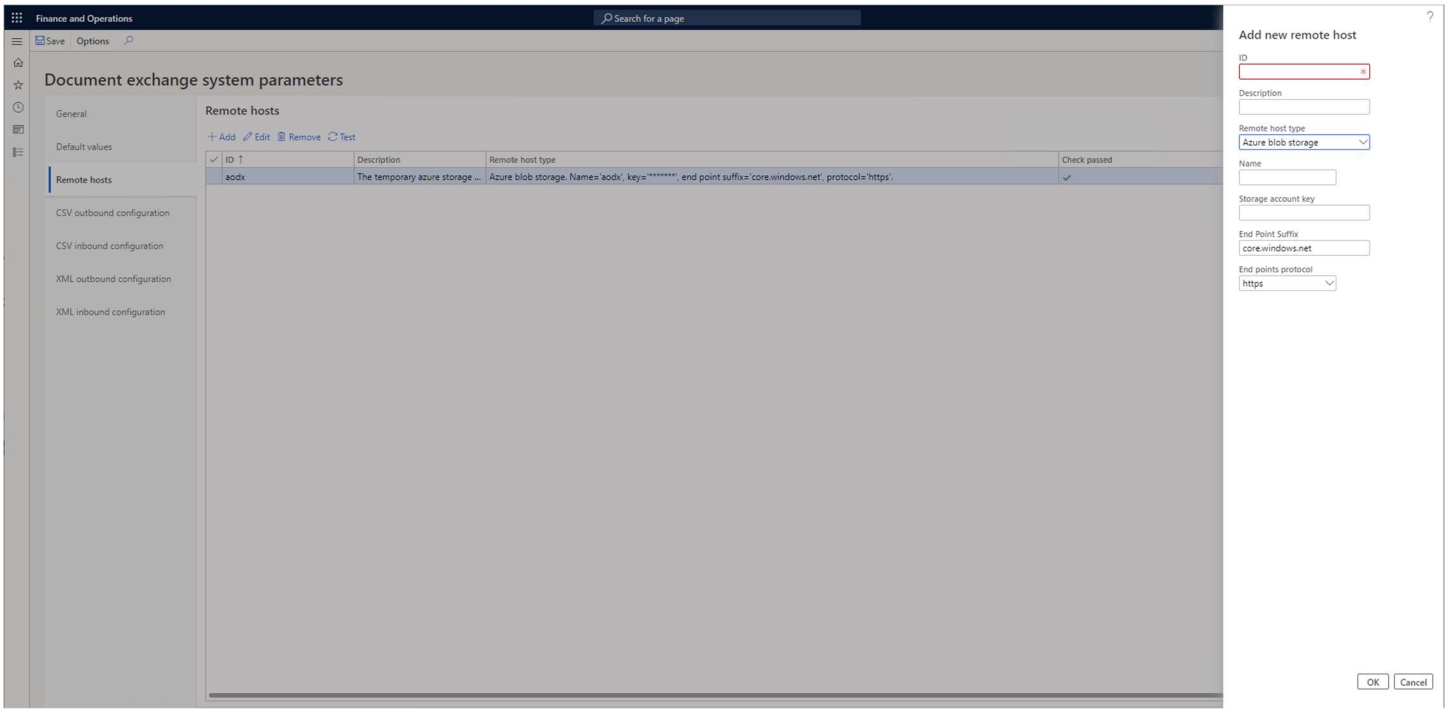
Host: The IP address or DNS name of the FTP or FTPS server (example: 56.7.8.9 or server.somedomain.com - required).

Username: The username part of the credentials (required).

Password: The password part of the credentials (required).

FTP settings: The FTP settings that must be configured for certain FTP and FTPS servers. Generally, the *Default* settings record works for most servers (required).

Add an Azure Storage account



Fill in these fields:

ID: Any alphanumeric identifier that will uniquely identify this remote host (required).

Description: A meaningful description of the remote host (optional).

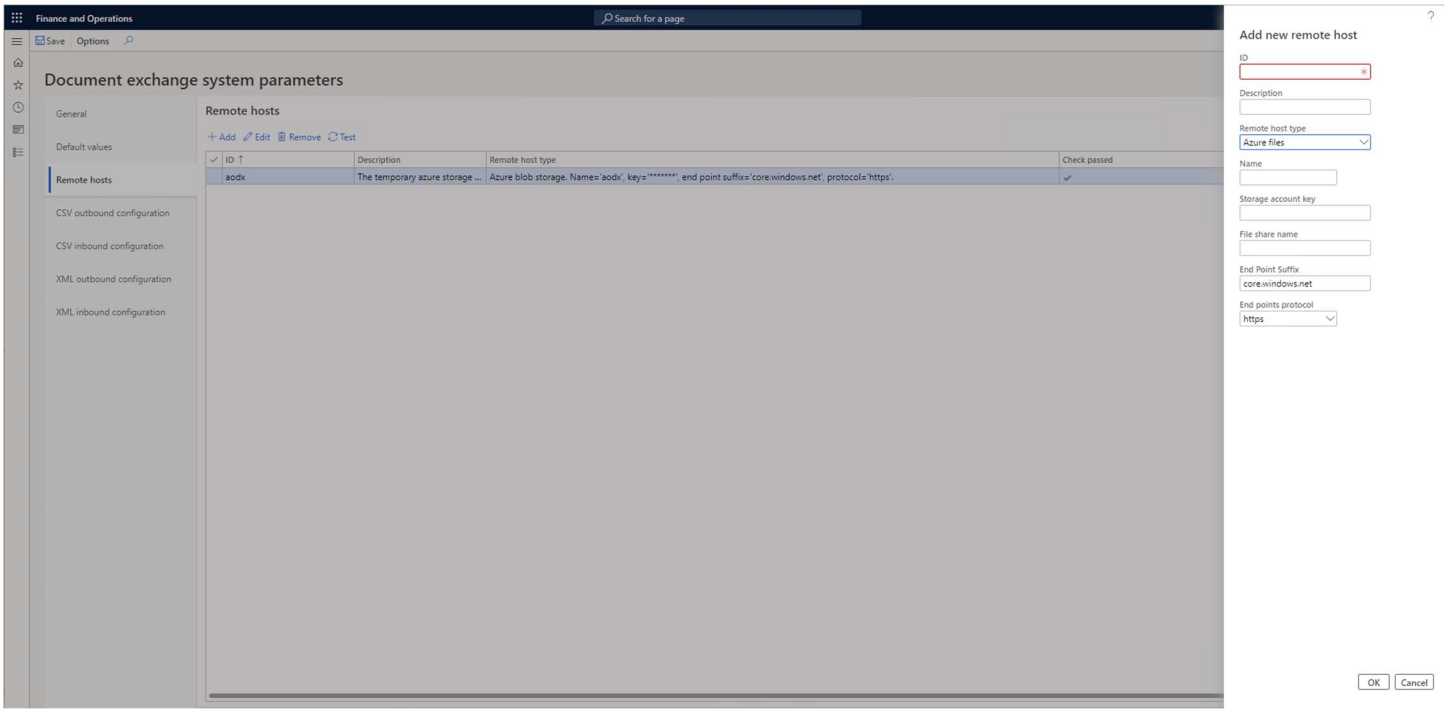
Name: The name of the Azure Storage account (required).

Storage account key: The storage account key is a credential that together with the account name prevents unauthorized access to the account (required).

End point suffix: The end point suffix that determines the Azure cloud region, it is generally core.windows.net or core.chinacloudapi.cn (required).

End points protocol: The protocol that the storage account uses to communicate (required). http (unsecure) or https (secure and encrypted).

Add an Azure Files share



Fill in these fields:

ID: Any alphanumeric identifier that will uniquely identify this remote host (required). In this case the *ID* matches the *Name* but it doesn't have to.

Description: A meaningful description of the remote host (optional).

Name: The name of the Azure Storage account (required).

Storage account key: The storage account key is a credential that together with the account name prevents unauthorized access to the account (required).

File share name: The name of the file share, must be a valid DNS name (required).

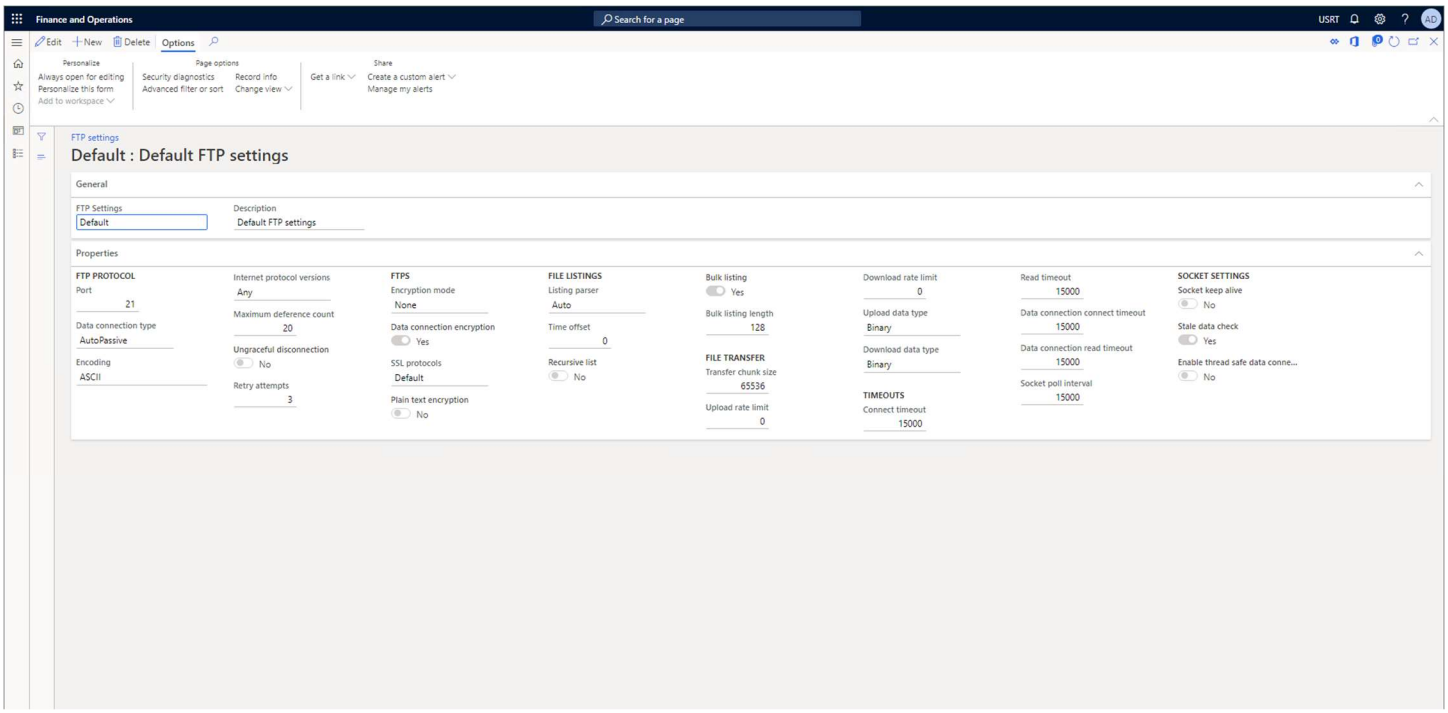
End point suffix: The end point suffix that determines the Azure cloud region, it is generally core.windows.net or core.chinacloudapi.cn (required).

End points protocol: The protocol that the storage account uses to communicate (required). http (unsecure) or https (secure and encrypted).

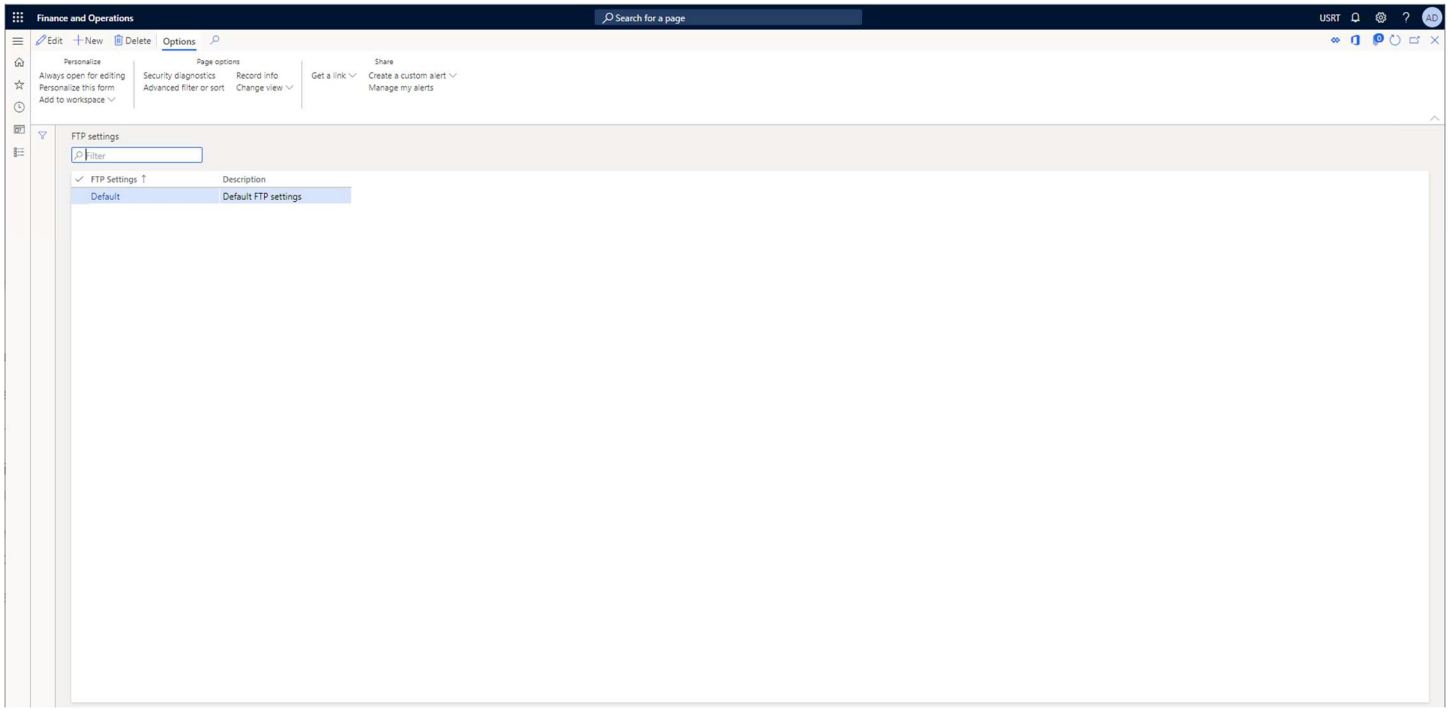
FTP Settings

FTP and FTPS servers have upwards of 25 different configuration properties that may affect connections to certain servers. To simplify setup and administration they have been grouped together, so that they can be reused for different remote hosts.

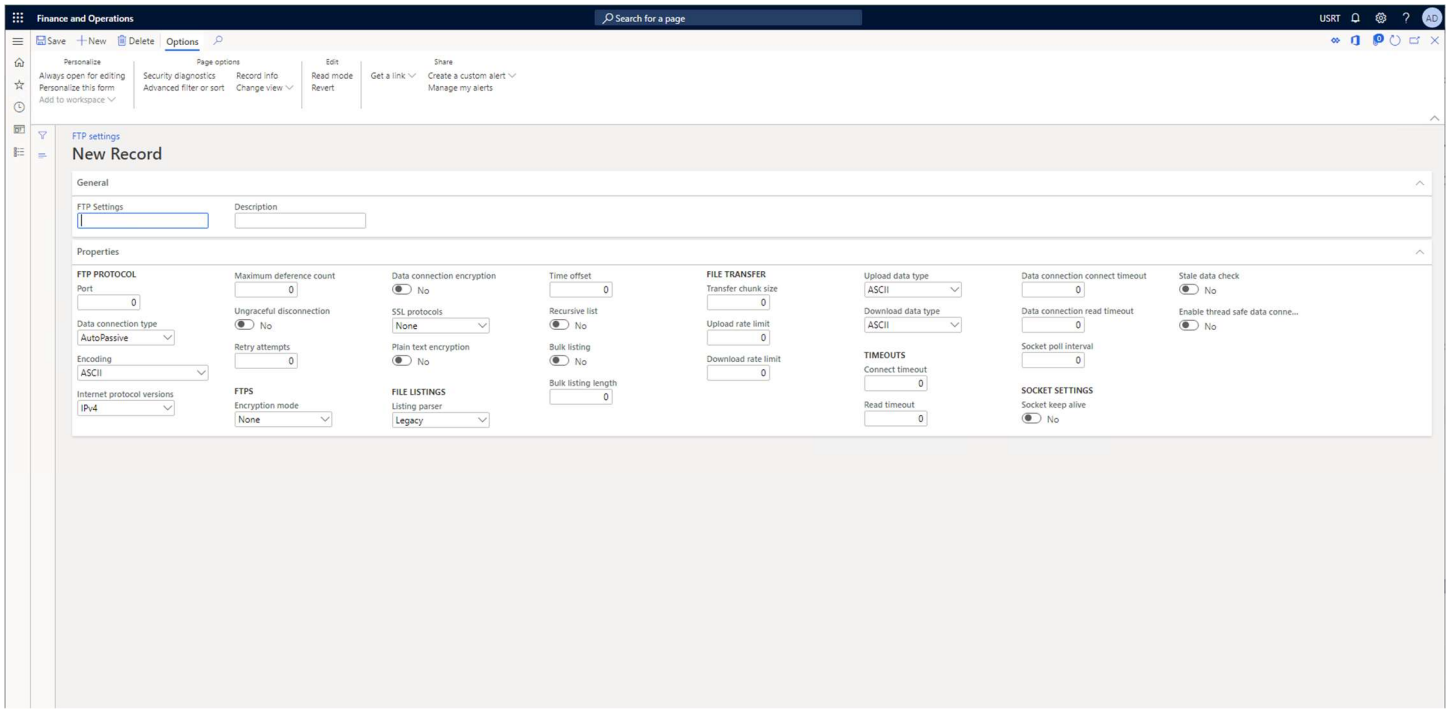
On initialization the AODX system creates a Default record for these configuration properties. This Default record contains standard values for these properties:



You can also create and customize your own FTP Settings records depending on your needs. To add a new FTP settings record go to: *Modules -> Document Exchange System -> Setup -> FTP settings*.



Click on the *New* button.



General.

FTP Settings: Any alphanumeric identifier that will uniquely identify the FTP settings record (required).

Description: A meaningful description of the FTP settings record(optional).

FTP protocol.

Port: The communications end point.

Data connection type: AutoPassive, PASV, PASVEX, EPSV, AutoActive, PORT or EPRT.

Encoding: ASCII, BigEndianUnicode, Default, UTF32, UTF7, UTF8 or Unicode. The text encoding option.

Internet protocol versions: IPv4, IPv6 or Any.

Maximum deference count: The maximum depth of recursion that the DereferenceLink command will follow before giving up.

Ungraceful disconnection: Yes or No. Disconnect from the server without sending the QUIT command.

Retry attempts: Number of times to retry when a verification failure occurs during an upload or a download.

FTPS.

Encryption mode: None, Implicit or Explicit.

Data connection encryption: Yes or No. Determines if the data channels should be encrypted.

SSL protocols: None, Ssl2, Ssl3, Tls, Default, Tls11 or Tls12.

Plain text encryption: Disable encryption immediately after connecting to the server.

File listings.

Listing parser: Legacy, Auto, Machine, Windows, Unix, UnixAlt, VMS, IBM or NonStop.

Time offset: Hour difference between server and client.

Recursive list: Yes or No. Check if the server supports the LIST command.

Bulk listing: Yes or No. If set to *Yes* increases the performance of an internal FluentFTP command called *GetListing* by reading multiple lines at once.

Bulk listing length: Bytes to read during the execution of the *GetListing* command. Only valid when the *Bulk listing* property is set to *Yes*.

File transfers.

Transfer chunk size: Size in bytes of a chunk during upload and download operations.

Upload rate limit: The rate limit for uploads in kilobytes.

Download rate limit: The rate limit for downloads in kilobytes.

Upload data type: ASCII or Binary.

Download data type: ASCII or Binary.

Timeouts.

Connect timeout: Maximum number of milliseconds to wait for a connection attempt to succeed.

Read timeout: Milliseconds to wait for data to be read from the stream.

Data connection connect timeout: Maximum number of milliseconds to wait until a successful connection can be established.

Data connection read timeout: Maximum number of milliseconds to wait when reading data.

Socket poll interval: Milliseconds that must elapse before calling Poll on the socket.

Socket Settings.

Socket keep alive: Yes or No. Keep the socket alive.

Stale data check: Yes or No. Check if there is stale data on the socket.

Enable thread safe data connections: Yes or No. Creates a new FTP connection for each file being uploaded or downloaded.

.

Documents and Versions

A Document in the ADOX system is a CSV or XML file than can be imported or exported. Different files will have different Documents. There are four basic Document types:

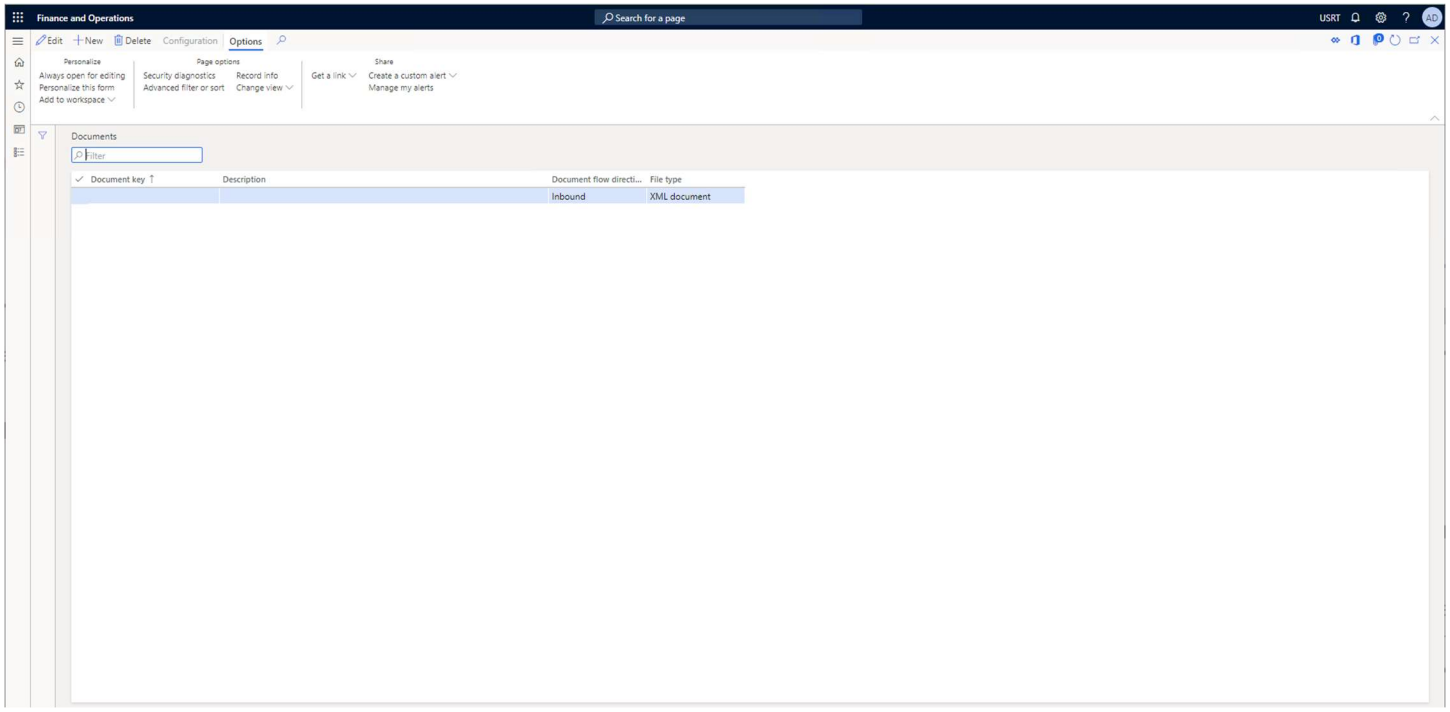
- CSV export or outbound file
- CSV import or inbound file
- XML export or outbound file
- XML import or inbound file

You must define the Document type when you create the record, this cannot be changed later. You cannot create a CSV outbound Document and then change it to an XML inbound Document.

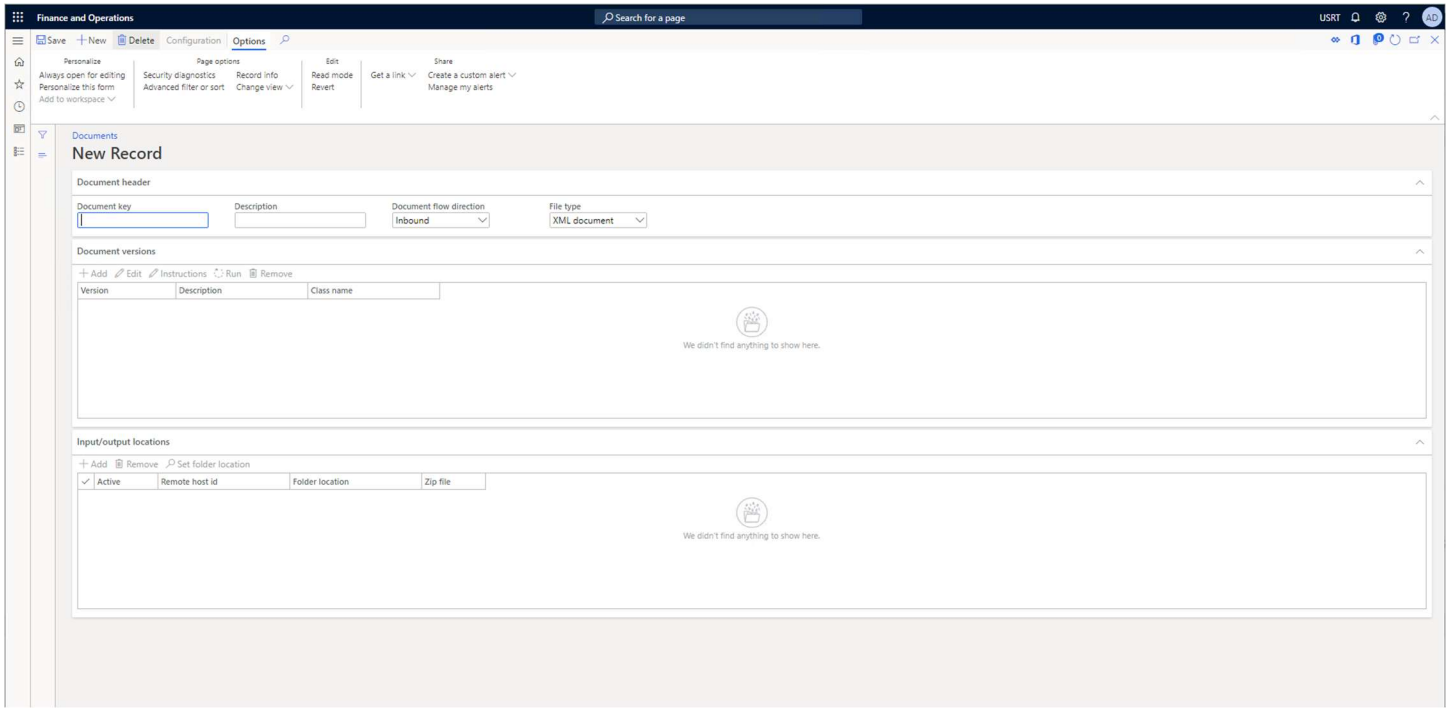
Since changes might occur to a Document over time, the AODX system also has the concept of Document Versions. You create Instructions of how to export or import a Document on the Document Version and not on the Document itself. This gives you the flexibility of creating new Document Versions to adapt to changes in Document specifications while having the ability to fall back on previous versions if needed.

All Documents are imported from and exported to Remote Hosts or external servers. In the Document definition you specify those remote hosts or servers. When exporting a Document you can send it to one or more Remote Hosts, creating several copies of the exported Document. When importing a Document you can only specify a single source Remote Host. For added flexibility you can override these input and output Remote Hosts on the Document Version and have different Remote Hosts for each Document Version.

To create a Document go to: *Modules -> Document Exchange System -> Documents and queries -> Documents*.



Click on the *New* button.



To create a Document fill in these fields:

Document key: Any alphanumeric identifier that will uniquely identify the Document (required).

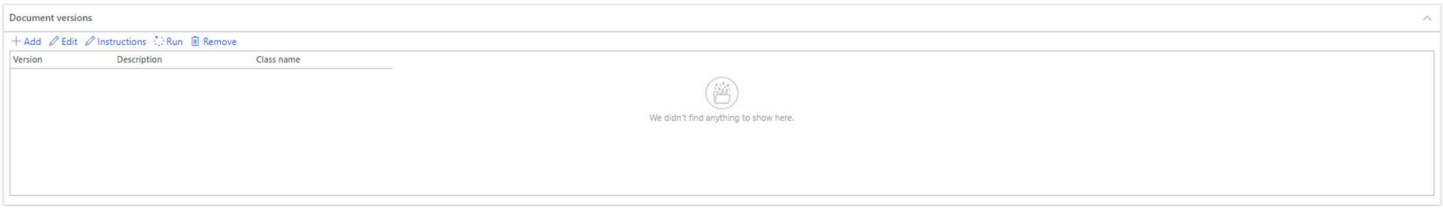
Description: A meaningful description of the Document (optional).

Document flow direction: Inbound or Outbound. Inbound is for files that are coming into the Dynamics 365 system or being imported. Outbound is for files that are going out from the Dynamics 365 system or being exported.

File type: XML Document or CSV Document.

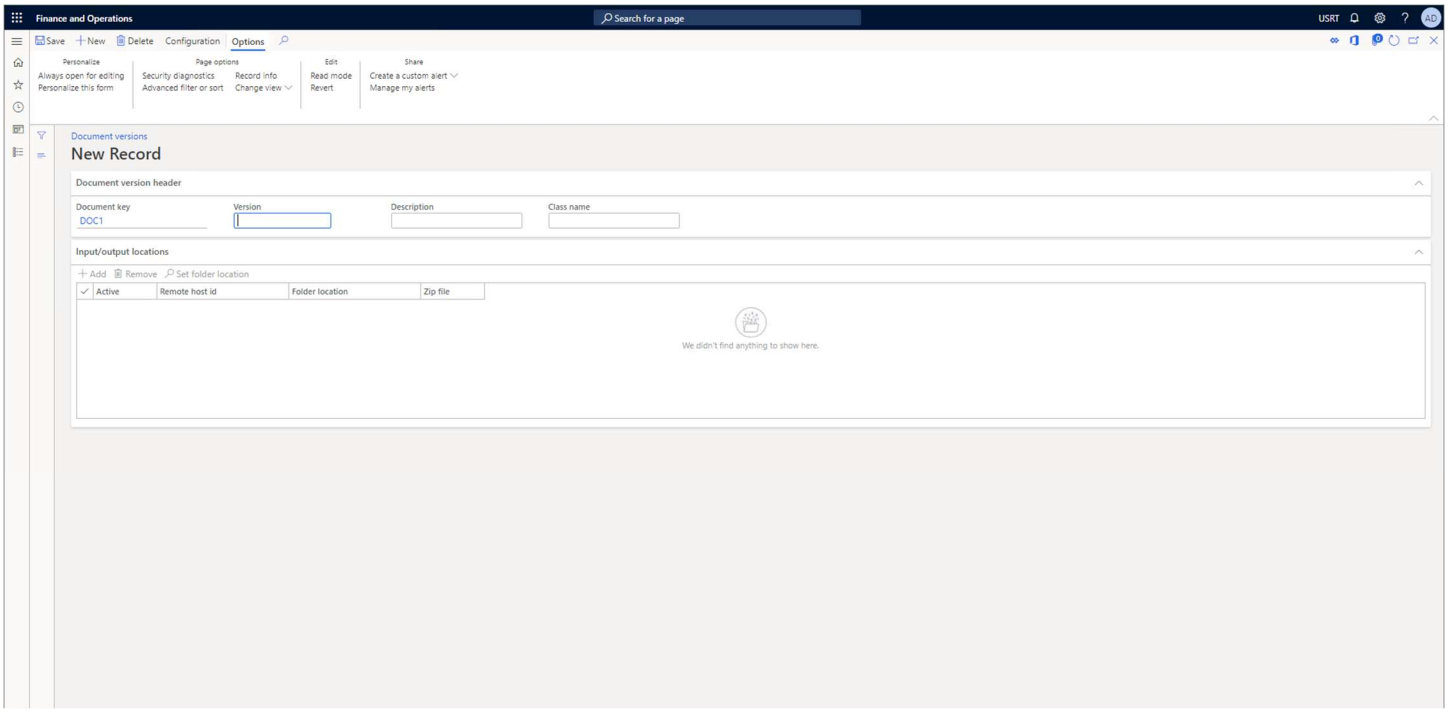
Once you click on the *Save* button, the *Document versions* grid and the *Input/output locations* grid will be enabled.

Document Versions.



On the *Document versions* grid on the *Documents* form you will Add, Edit, Edit Instructions, Run (execute) and Remove **Document Versions**.

When you click on the *Add* button you will be taken to the Document Versions form:



To create a Document Version, you must fill in these fields:

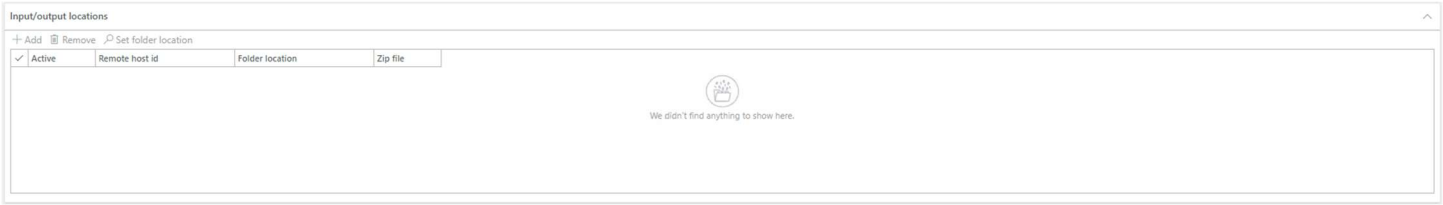
Version: Any alphanumeric identifier that will uniquely identify the Document Version (required). It can contain letters, but a simple version number like 1.0.0 is recommended.

Description: A meaningful description of the Document Version (optional).

Class name: The name of the class that extends this Document Version. Only required when the Document Version's Instructions have been hard coded in X++ in an extension class.

Instructions on each Document Version will vary for each Document type so they will be covered

Input/Output Locations.



On the *Input/output locations* grid on the *Documents* form or the *Document Versions* form you will Add, Remove and Set folder location. When you click on the *Add* button you must fill in these fields:

Active: Is the location active? You can mark a location as inactive, so you do not have to remove it.

Remote host id: The ID of the Remote Host that can be selected via a dropdown.

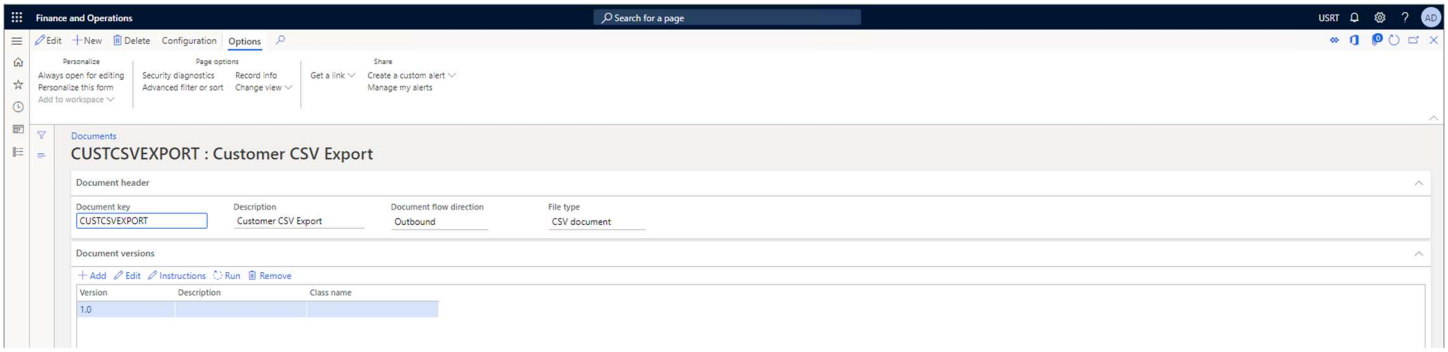
Folder location: The path of the folder on the Remote Host. Can only be selected by using the Set folder location button.

Zip file: Unzip or zip? Are the incoming files inside a zip file or do you want the outgoing file to be zipped.

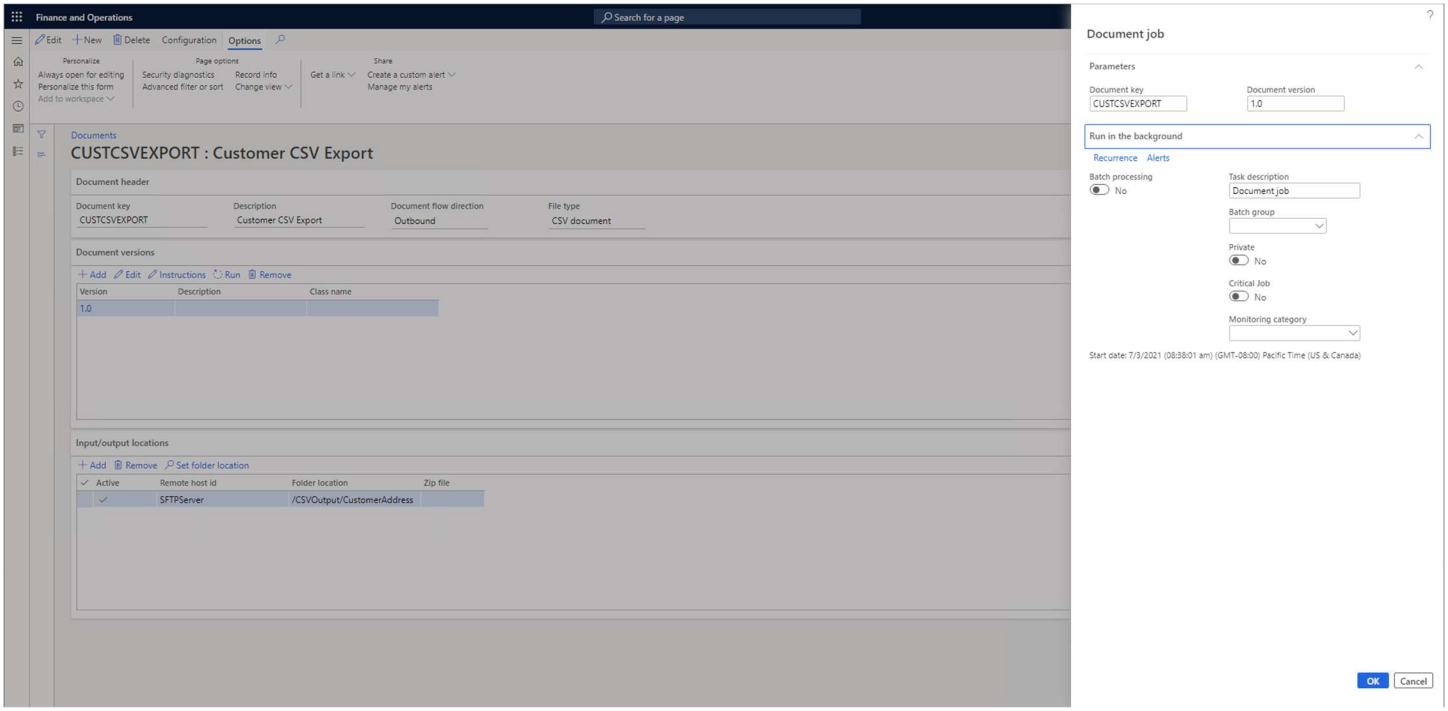
The *input/output locations* grid on the Document Versions form will override any setting on the Documents Form.

Execute a Document Version and Setup Recurring integrations

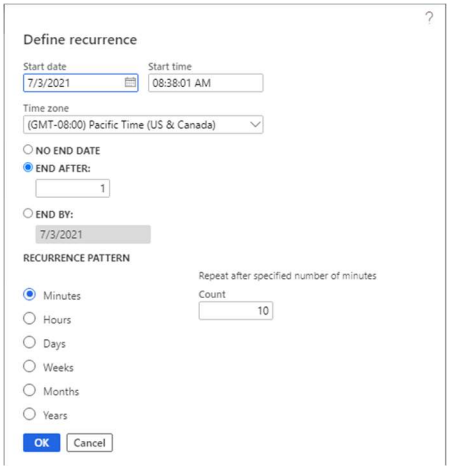
Once your Document Version is setup correctly go to the Document versions grid on the Documents form, select any Document Version and click on the *Run* button to execute it.



You will get a prompt like this one:



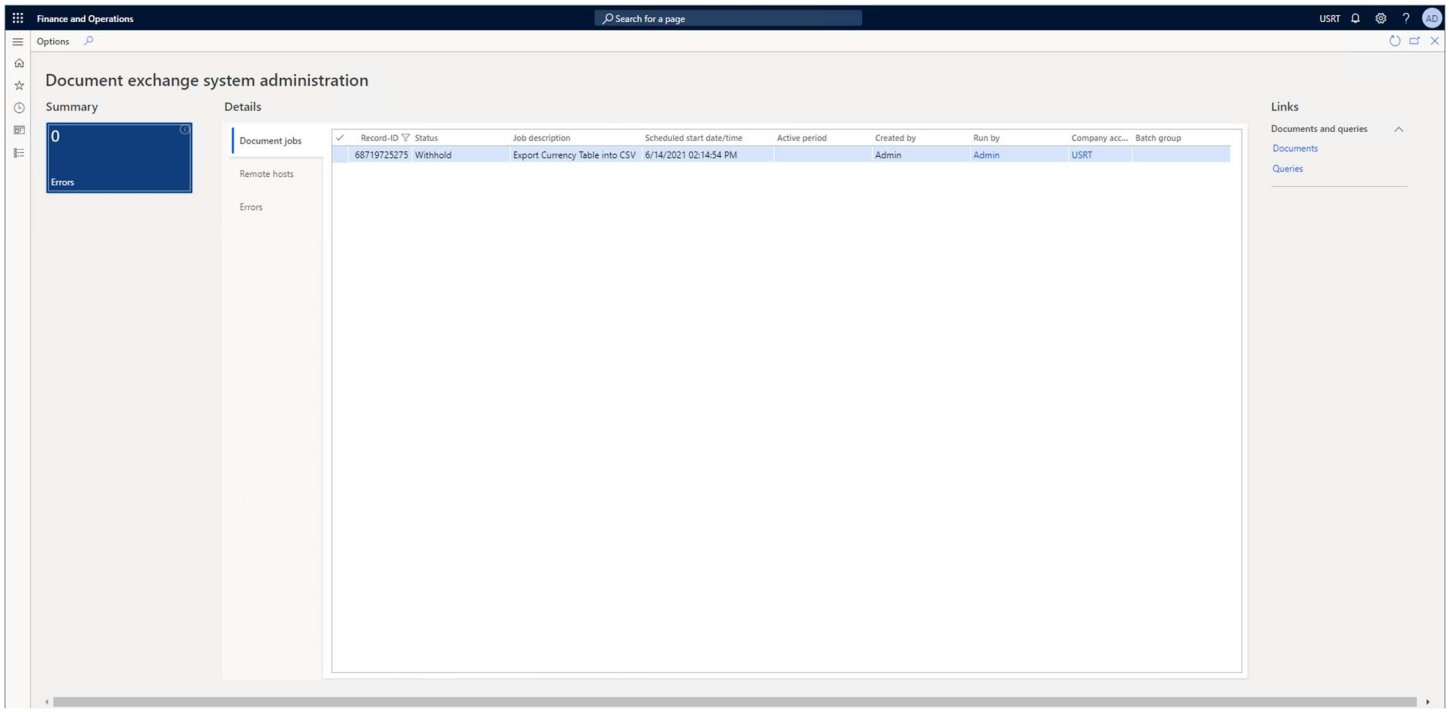
If you set *Batch processing* to *Yes,* the Document Version integration will run offline via the batch framework in Dynamics 365. The D365 system will create a batch job. You will also be able to set a recurrence on it.



If you set *Batch Processing* to *No*, the Document Version integration will run a single time and will do so immediately.

Administration Workspace

To open the Administration Workspace, go to: *Modules -> Document Exchange System -> Workspaces -> Document exchange system administration*.



The Administration Workspace has three tabs.

In the *Document Jobs* tab, you will be able to check on the status of the recurring jobs that you have set up for the AODX System. It is filtered and only contains AODX system jobs.

The *Remote hosts* tab will show you all the Remote Hosts that you have configured for the system. You will also be able to navigate the directory structure of these Remote Hosts and visualize files without having to leave the D365 environment.

The *Errors tab* will show you the errors that the AODX system has reported.

Export System General Overview

Export a CSV Document.

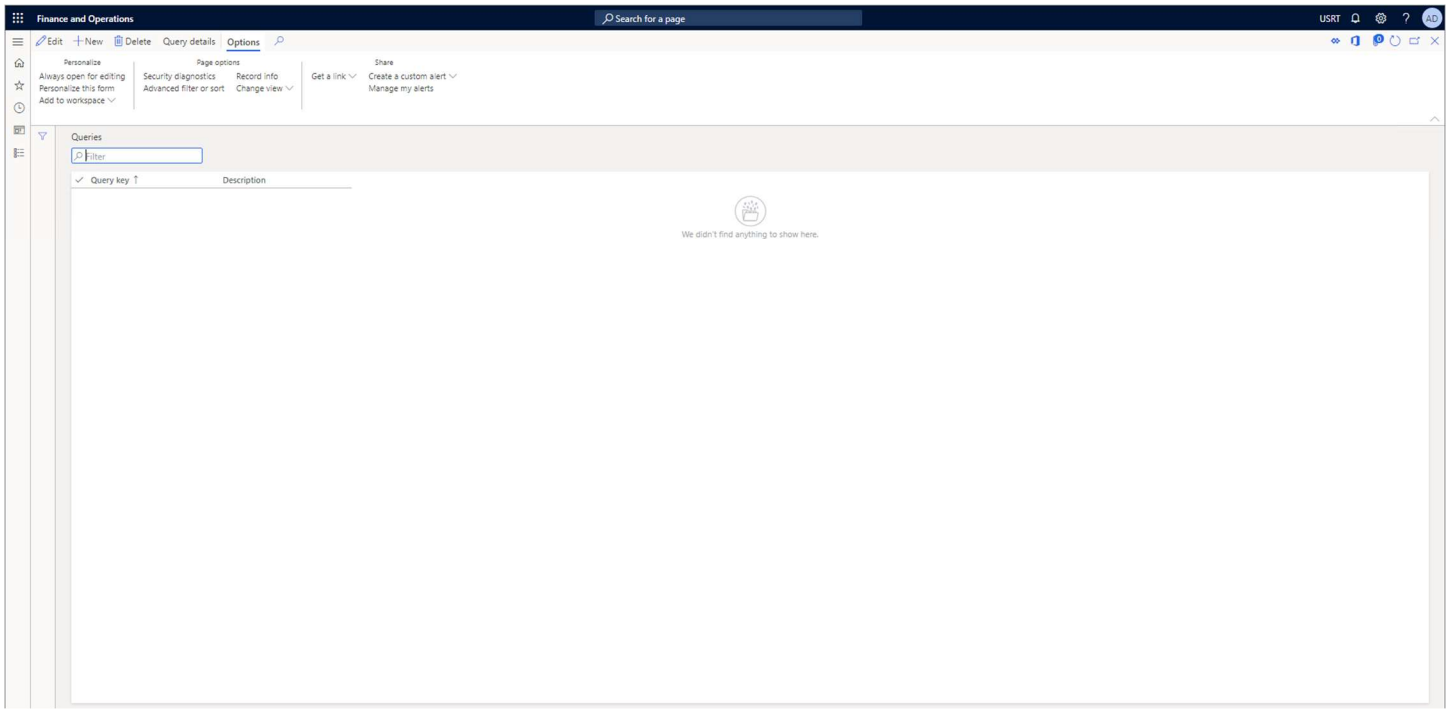
To export a CSV document, you must define a Query first. This Query will contain the fields and records that you want to export. A Query can contain one or more tables. Once you have designed a Query you must create a Document and a Document Version. When you have defined a Document Version you create a set of Instructions that determine how the Query is transformed into a CSV document. After the Instructions are defined you execute the Document Version. When you execute the Document Version, the CSV document is created in the temporary Azure Storage container and then copied over to an SFTP, FTP, FTPS, Azure Storage or Azure Files server.

Export an XML Document.

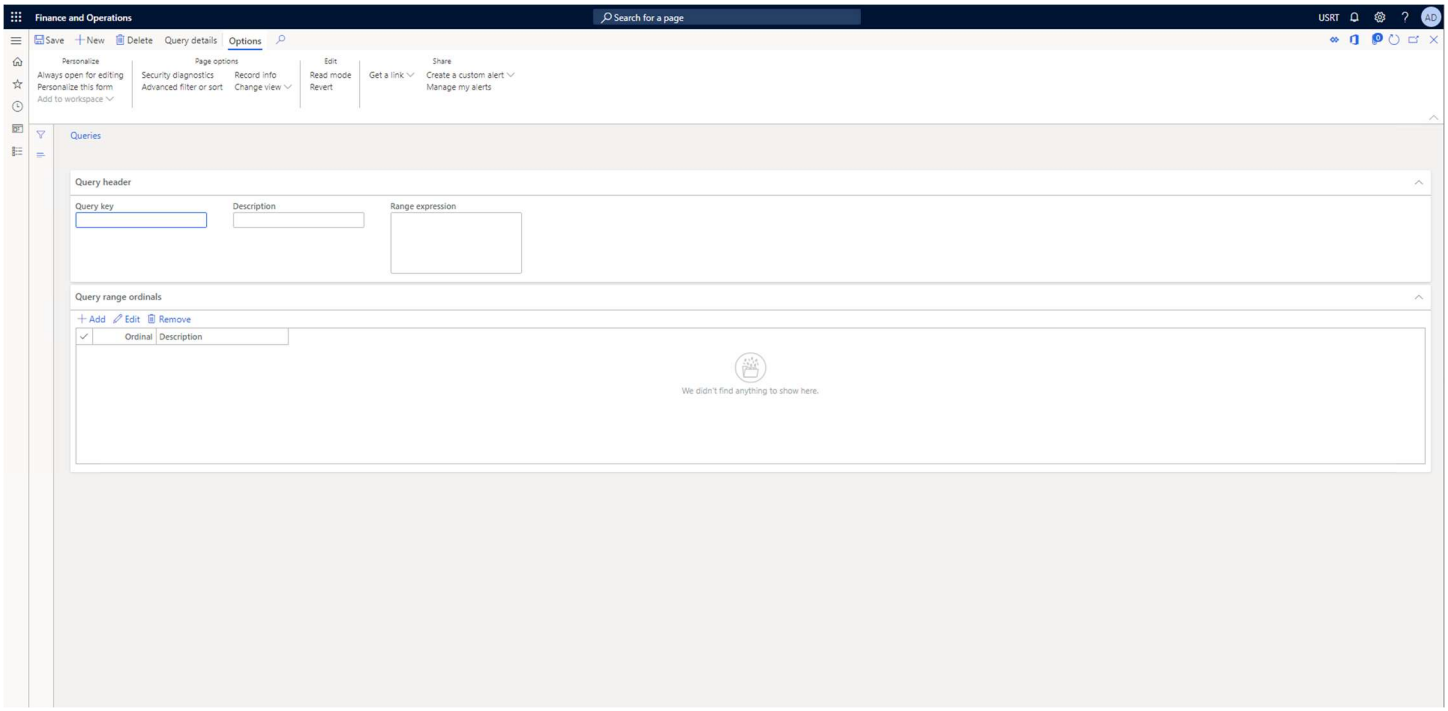
To export an XML Document, you first must define at least one Query. Because of its hierarchical nature an XML document can have more than one Query. Once you have designed the Query or Queries you must create a Document and a Document Version. When you have defined a Document Version you create a set of hierarchical Instructions that determine how the Query or Queries are transformed into an XML document. After the Instructions are defined you execute the Document Version. When you execute the Document Version, the XML document is created in the temporary Azure Storage container and then copied over to an SFTP, FTP, FTPS, Azure Storage or Azure Files Server.

Queries

To export a CSV or an XML Document, you must first create a Query. To create a Query go to: *Modules -> Document Exchange System -> Documents and queries -> Queries*



To create a Query click on the *New* button.



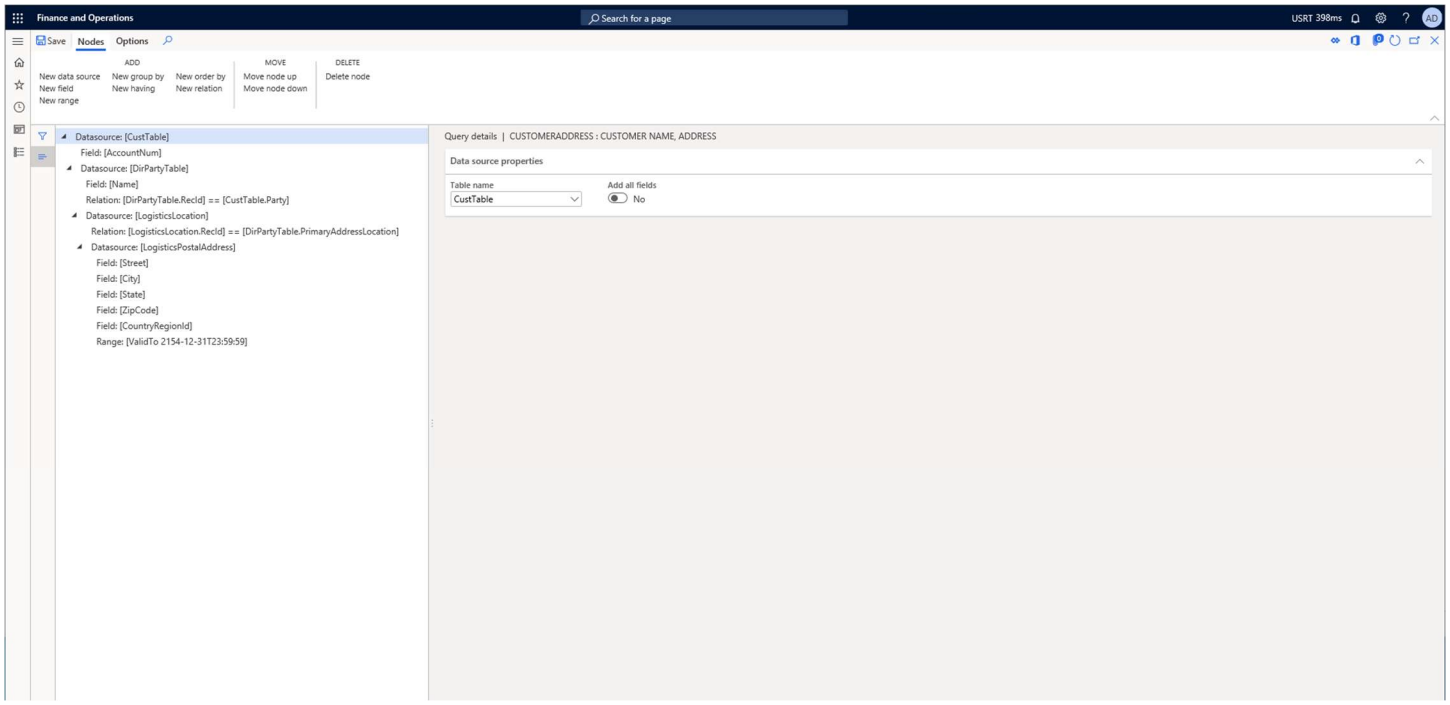
Query key: Any alphanumeric identifier that will uniquely identify the Query (required).

Description: A meaningful description of the Query (optional).

Range expression: A range expression can be entered here and in conjunction with the values entered in the *Query range ordinals* grid will allow the Query to have OR conditions (optional).

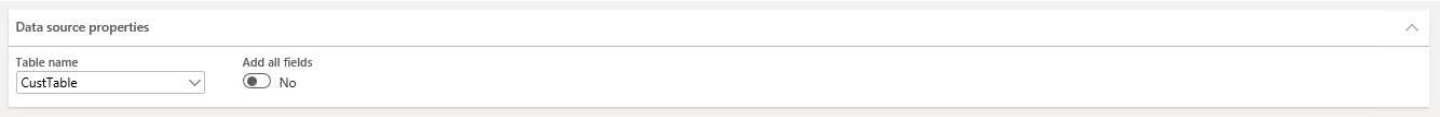
Query Details.

Click on the *Query details* button in the *Queries* form to open the *Query details* form where you can edit the Query objects. A Query is a hierarchical tree of items. The root item is a Datasource which represents a table.

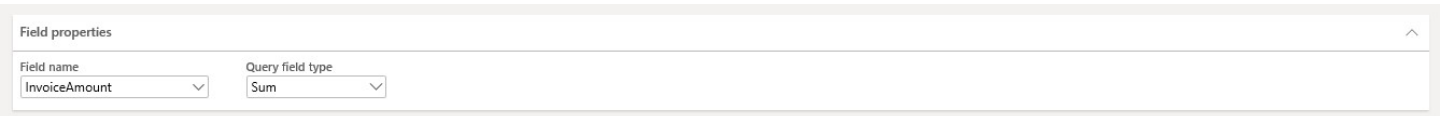


A Query has the following types of items:

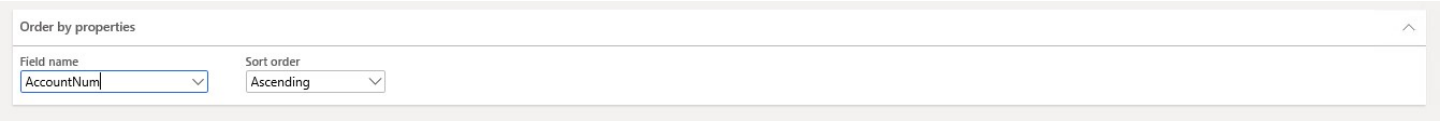
Datasource: Represents a table in the database. A Datasource is the only object that can have children objects. You have the option of setting the *Add all fields* radio box to Yes and all the fields in that table will be added to the Query or specifying each field individually as a child object. Datasource items can be child items of other Datasources and together with the Relations item you determine how tables are linked in the Query.



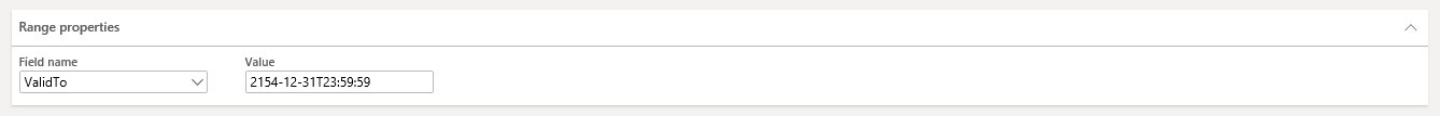
Field: Represents a field from a table. It can be a standard field or an aggregate calculation such as: Average, Sum, Minimum, Maximum or Count.



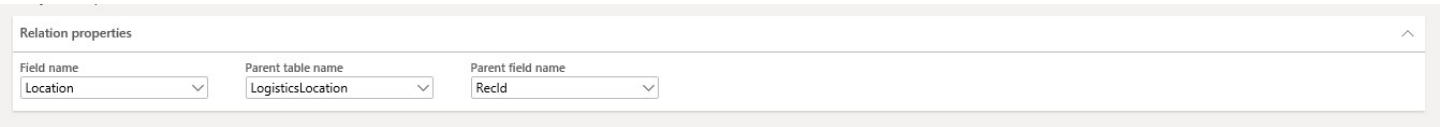
Sort by: This object establishes the order in which the fields are listed in the Query. You can specify an Ascending or Descending order.



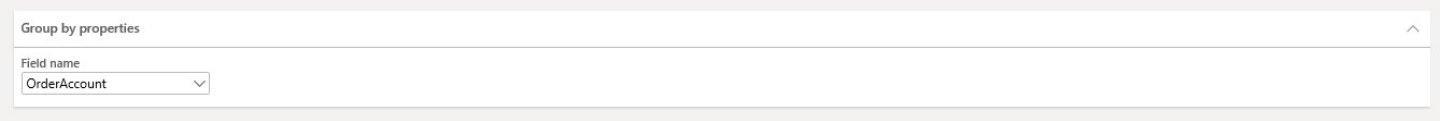
Range: Represents a field in a SQL where condition. When several ranges exist on a Datasource they are combined using the AND operator.



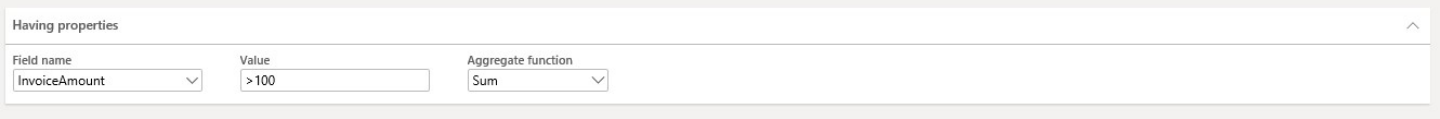
Relation: Represents a relation between a field in a Datasource and another Field in a parent Datasource.



Group by: Represents the SQL group by clause. The group by clause is used to group and summarize similar data and is used in aggregate queries.



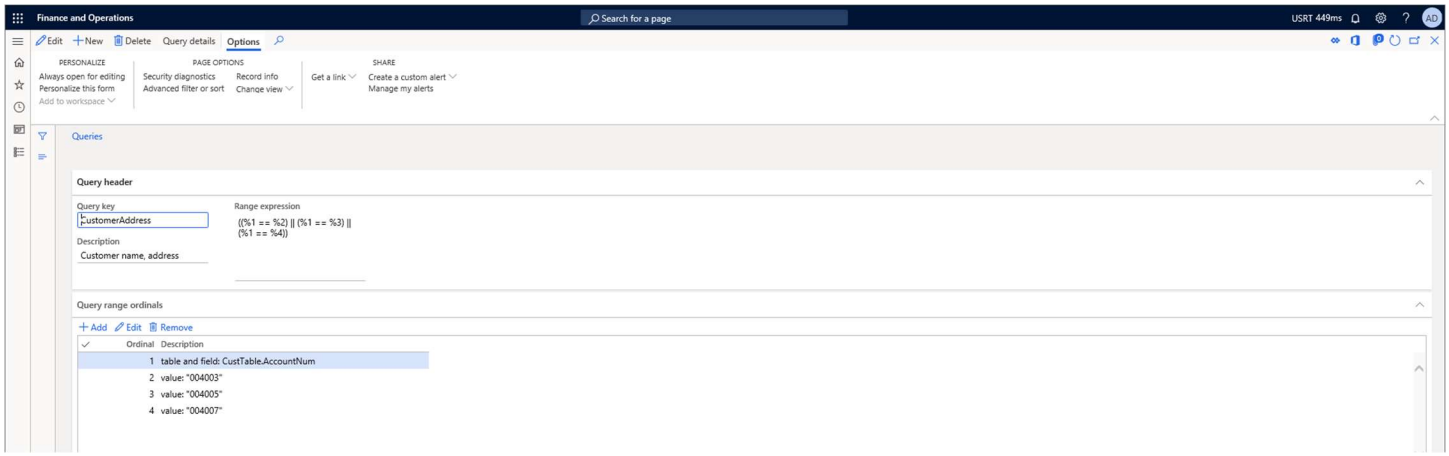
Having: Represents the SQL having clause. A having clause is equal to the SQL where condition but uses aggregate functions such as: Average, Sum, Minimum, Maximum, and Count.



Query Range Ordinals.

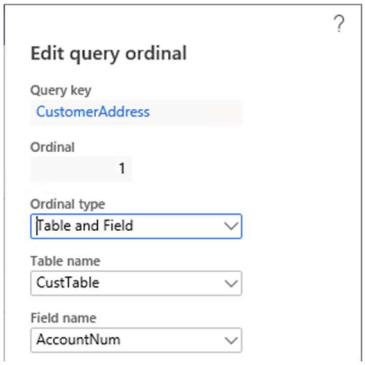
Writing complex WHERE conditions can be hard or even impossible using only range syntax. The Query range ordinals grid in conjunction with the Range expression field gives you an additional level of flexibility when you must write complex WHERE conditions on the Query object.

You can write any type of WHERE condition on the Range expression field with placeholders with a percentage sign (%) prefix, and the values specified in the Query range ordinals grid will replace these placeholders at runtime.

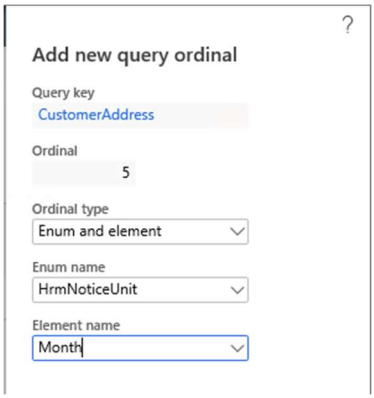


Query range ordinals have three types: a table/field, an enumeration type/element or a value.

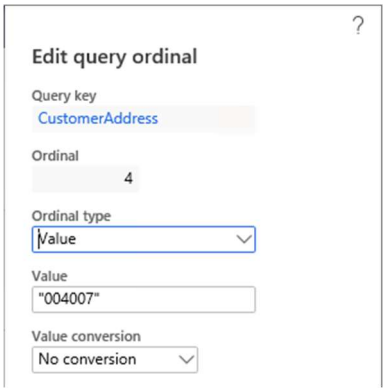
A table/field pair:



An enumeration type/element pair:



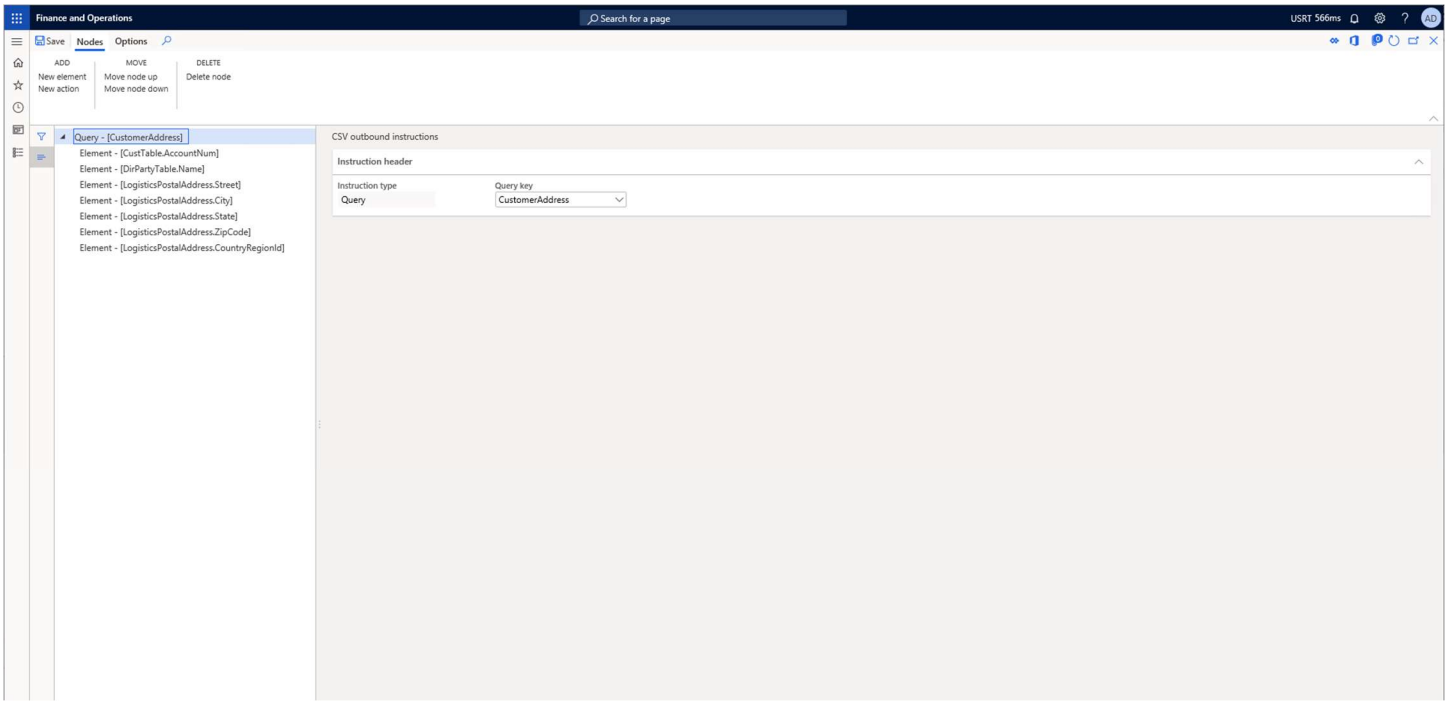
Or a value:



Since certain values sometimes must be escaped properly there is a Value conversion control. The default is *No conversion*, but you also have the option of using the *queryValue* function to convert your string.

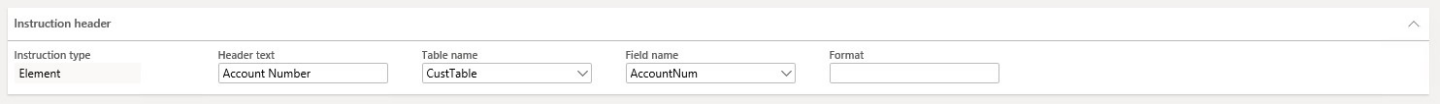
CSV Document Export

The instructions for an outbound CSV file are specified in a hierarchical tree style. The root element is a pointer to a Query object and the children are either Elements or Actions. The whole tree represents a single line on the output CSV file.

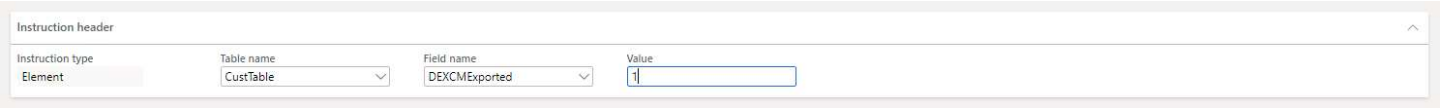


Instructions for an outbound CSV document that uses the CustomerAddress Query. These instructions will list the account number, customer name, address, city, state, postal code, and country on each line of the output CSV file.

An Element child item is the equivalent of a table/field pair from the parent Query. Element items of certain types such as dates or real numbers can have an associated Format property that follows the formatting for .NET custom date and time format strings or .NET custom numeric formats.



When the Query executes, Action child items change values on table fields on the Query. Action child items can be used to mark records that have been exported previously to allow incremental exports.



Customer Address CSV Export Example

This section will walk you through the creation of an output CSV file with customer account, name, and address information like this one:

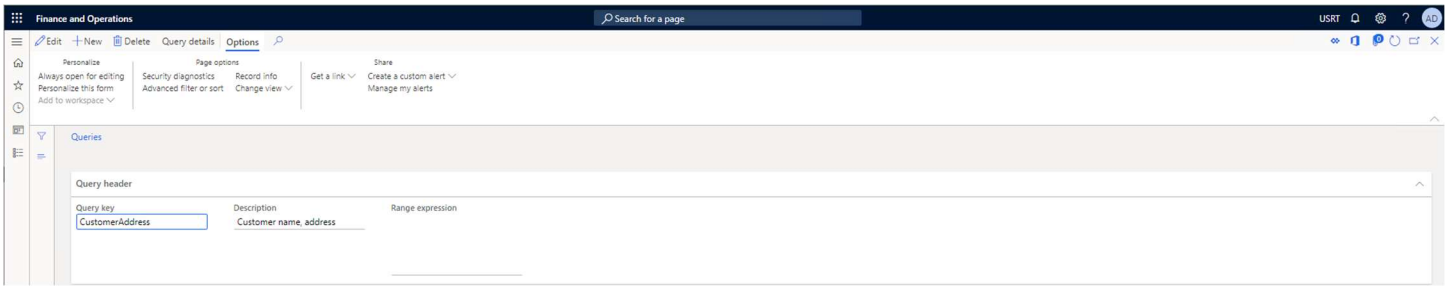
```
Account Number,Name,Address,City,State,Postal Code,Country
"004003","Mara Gentry","456 Ash Street","Oakland","CA","94115","USA"
"004005","Eve Whitehead","123 Oak Street","Redmond","WA","98007","USA"
"004007","Owen  Tolley","456 Sugar Hill","Tampa","FL","33601","USA"
"004009","Mathew Tolley","456 First Avenue","Alameda","CA","94115","USA"
"004011","Jennifer Beach","678 South 21st","Redmond","WA","98007","USA"
"004013","Shelly Beach","123 South Oak St","Renton","WA","98115","USA"
```

Dynamics 365 keeps basic customer information such as name, account number and address in four different tables. Those tables and their relations can better be described by looking at this Transact SQL statement from the AxDB database:

```
SELECT CT.ACCOUNTNUM, DPT.NAME, LPA.STREET, LPA.CITY, LPA.STATE, LPA.ZIPCODE, LPA.COUNTRYREGIONID FROM
CUSTTABLE AS CT
INNER JOIN DIRPARTYTABLE AS DPT ON CT.PARTY = DPT.RECID
INNER JOIN LOGISTICSLOCATION AS LL ON DPT.PRIMARYADDRESSLOCATION = LL.RECID
INNER JOIN LOGISTICSPOSTALADDRESS AS LPA ON LPA.LOCATION = LL.RECID
WHERE LPA.VALIDTO = '2154-12-31 23:59:59.000' AND CT.DATAAREAID = 'USRT'
ORDER BY CT.ACCOUNTNUM;
```

To create the output CSV file, you must first create a Query, go to: *Modules -> Document Exchange System -> Documents and queries -> Queries.*

Click on the *New* button.



Set the *Query key* control to *CustomerAddress*.

Set the *Description* control to *Customer name, address* (optional).

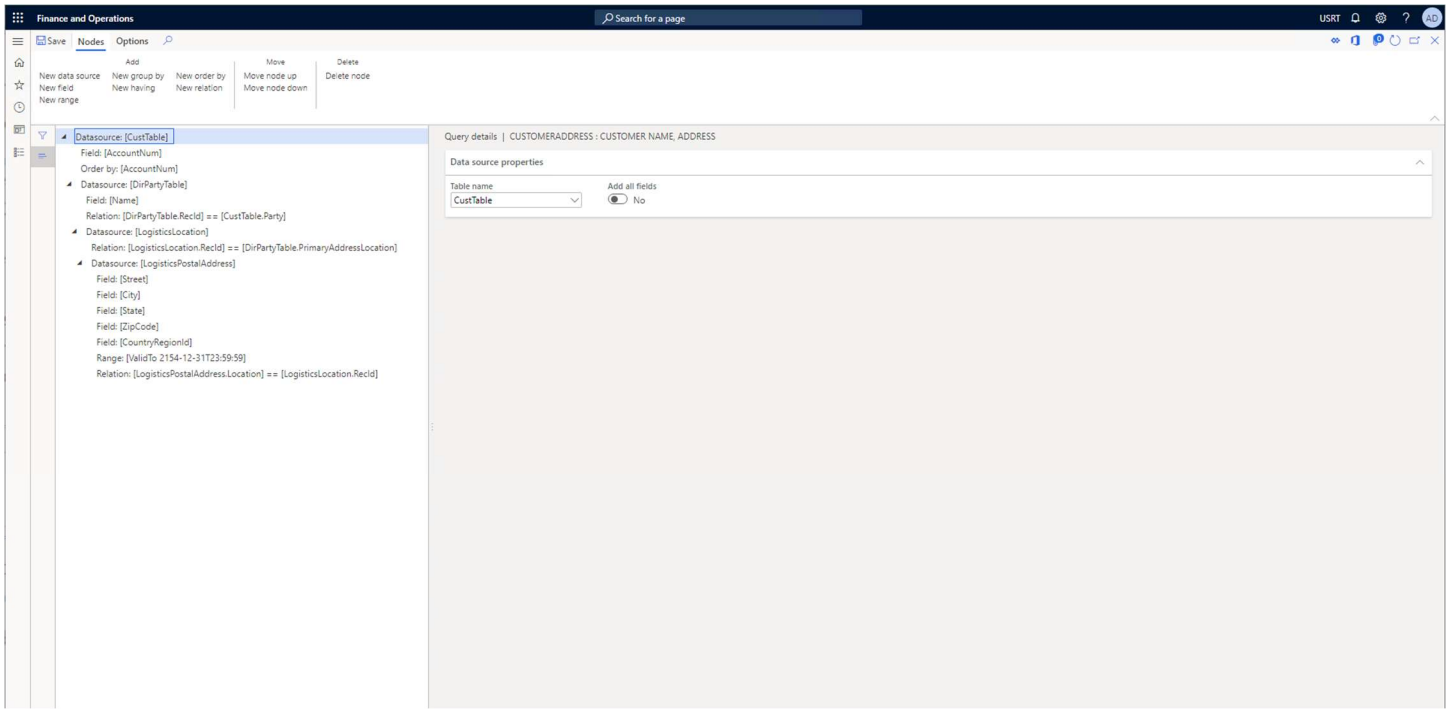
Click on the *Save* button.

Click on the *Query details* button.

Fill in the tree in the *Query details* form with these values:

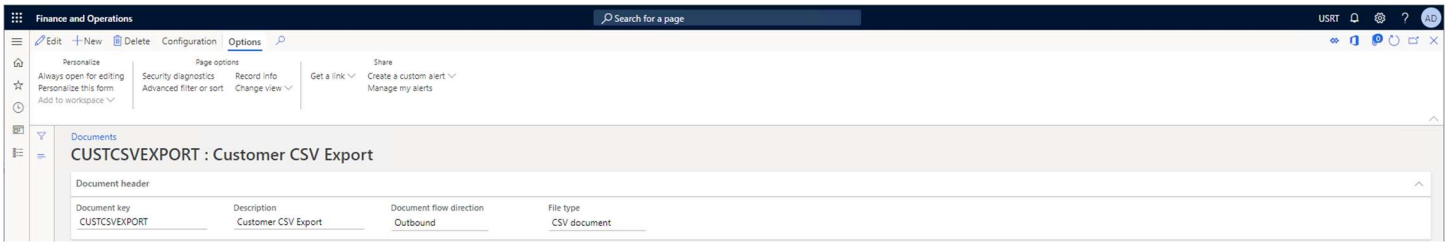
Datasource. *Table name:* CustTable, *Add all fields*: No.
- **Field.** *Field name:* AccountNum, *Query field type:* Standard.
- **Order by.** *Field name:* AccountNum
- **Datasource.** *Table name:* DirPartyTable, *Add all fields*: No.
  - **Field.** *Field name:* Name, *Query field type:* Standard.
  - **Relation.** *Field Name:* RecId, *Parent table name:* CustTable, *Parent field name:* Party.
  - **Datasource.** *Table name:* LogisticsLocation, *Add all fields:* No.
    - **Relation.** *Field Name:* RecId, *Parent table name:* DirPartyTable, *Parent field name:* PrimaryAddressLocation.
    - **Datasource.** *Table name:* LogisticsPostalAddress, *Add all fields:* No.
      - **Field.** *Field name:* Street, *Query field type:* Standard.
      - **Field.** *Field name:* City, *Query field type:* Standard.
      - **Field.** *Field name:* State, *Query field type:* Standard.
      - **Field.** *Field name:* ZipCode, *Query field type:* Standard.
      - **Field.** *Field name:* CountryRegionId, *Query field type:* Standard.
      - **Range.** *Field name:* ValidTo, *Value:* 2154-12-31T23:59:59.
      - **Relation.** *Field Name:* Location, *Parent table name:* LogisticsLocation, *Parent field name:* RecId.

After you have successfully entered all the above values your *Query details* tree should look like this:



Now that the Query is completed it is time to create a Document and a Document Version, go to: *Modules -> Document Exchange System -> Documents and queries -> Documents.*

Click on the *New* button.



Set the *Document Key* control to *CUSTCSVEXPORT*.

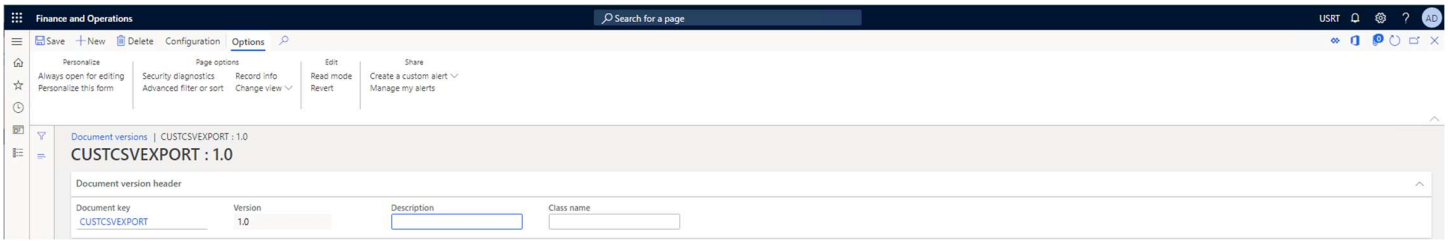
Set the *Description* control to *Customer CSV Export* (optional).

Set the *Document flow direction* control to *Outbound*.

Set the *File type* control to *CSV Document*.

Click on the *Save* button.

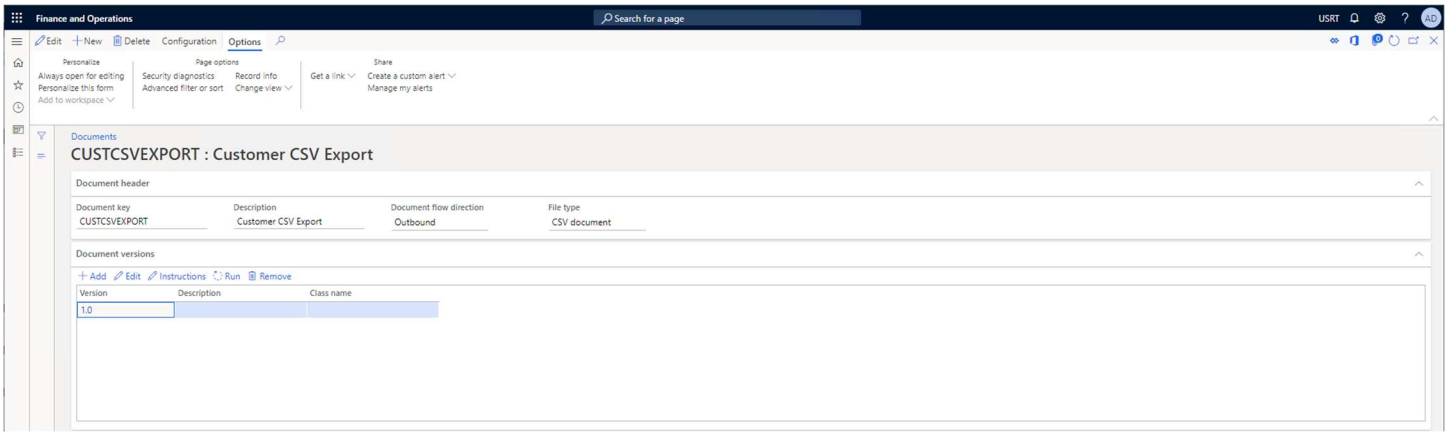
Click on the *Add* button in the *Document versions* grid to create a Document Version.



Set the *Version* control to *1.0*.

Exit the *Document versions* form.

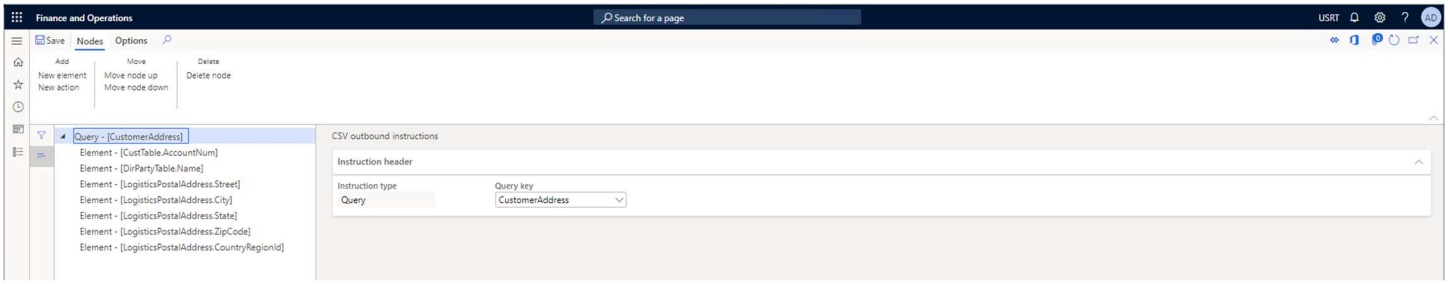
Click on the *Instructions* button in the *Document versions* grid.



Fill in the tree in the *CSV outbound instructions* form with these values:

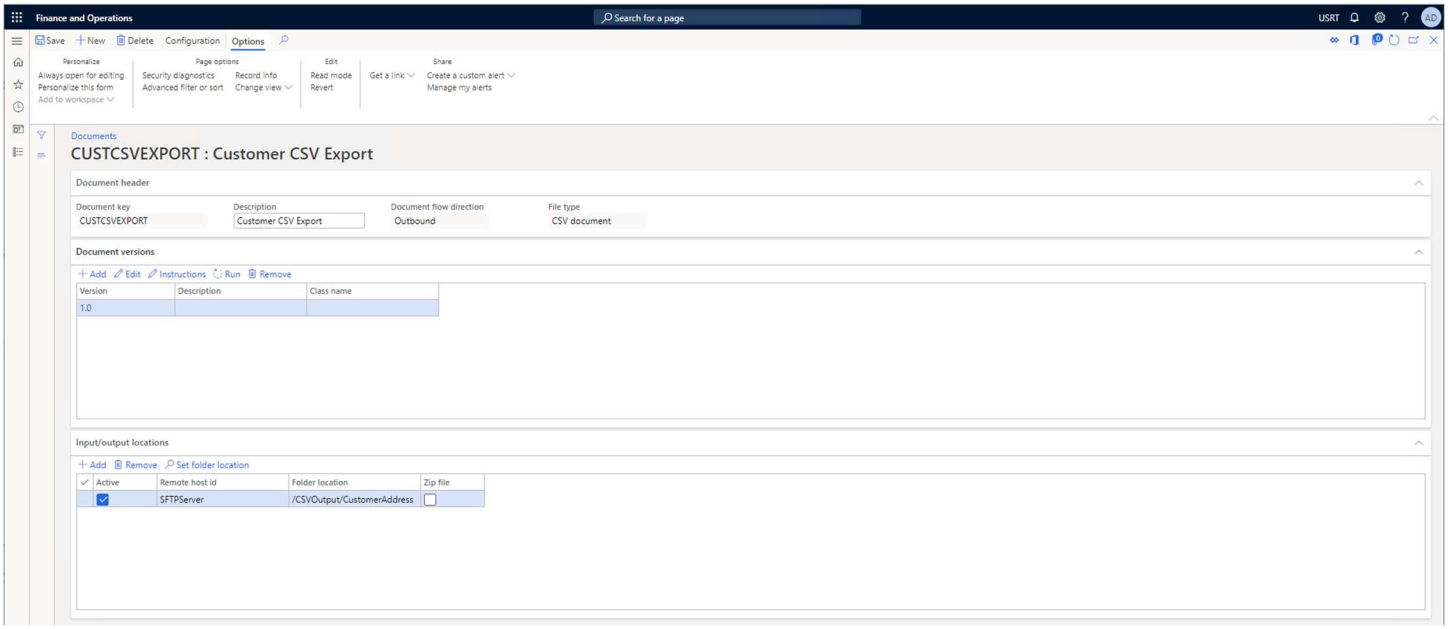
Query. *Query key:* CustomerAddress.
- **Element.** *Header text:* Account Number, *Table name:* CustTable, *Field name:* AccountNum.
- **Element.** *Header text:* Name, *Table name:* DirPartyTable, *Field name:* Name.
- **Element.** *Header text:* Address, *Table name:* LogisticsPostalAddress, *Field name:* Street.
- **Element.** *Header text:* City, *Table name:* LogisticsPostalAddress, *Field name:* City.
- **Element.** *Header text:* State, *Table name:* LogisticsPostalAddress, *Field name:* State.
- **Element.** *Header text:* Postal Code, *Table name:* LogisticsPostalAddress, *Field name:* ZipCode.
- **Element.** *Header text:* Country, *Table name:* LogisticsPostalAddress, *Field name:* CountryRegionId.

After you have successfully entered all the above values your *CSV outbound instructions* tree should look like this:



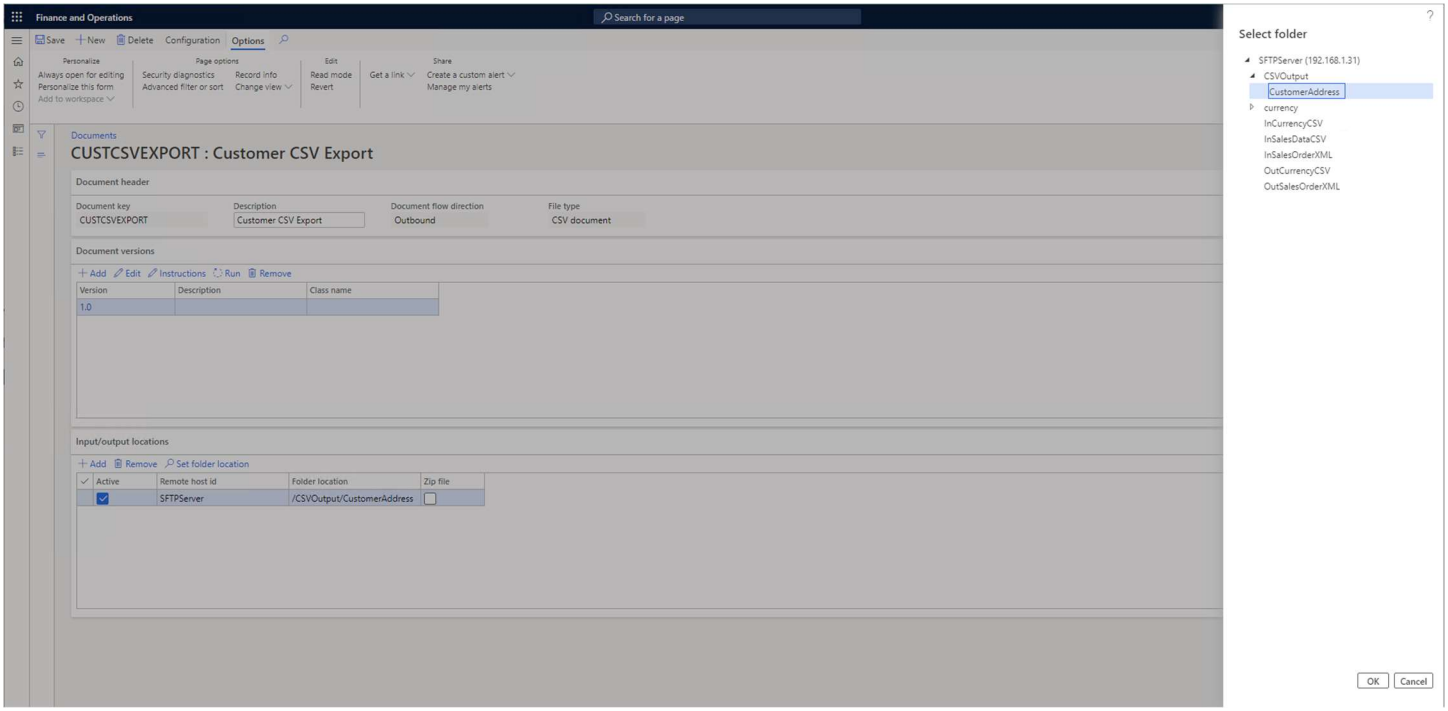
Exit the *CSV outbound instructions* form.

In the *Input/output locations* grid you must add a Remote host, set that Remote host's *Folder location* and set that Remote host to *Active*. This will vary according to the setup on your system.

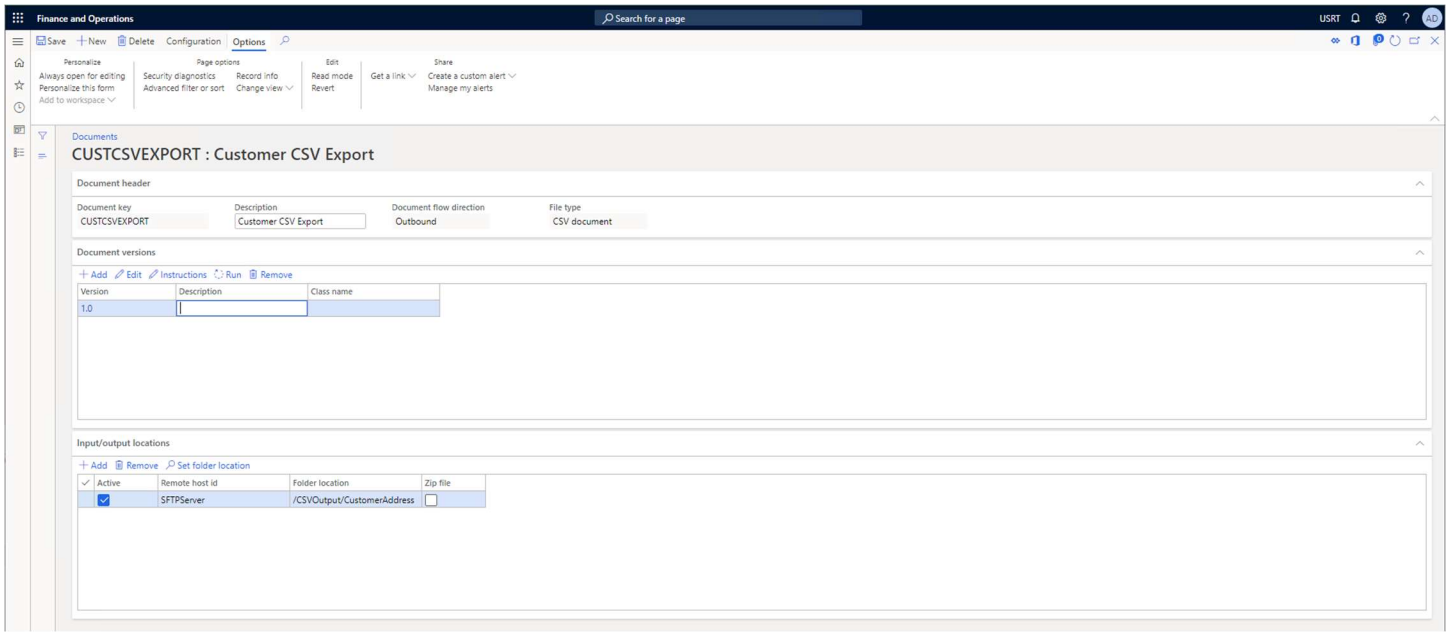


To set the *Folder location* use the *Set folder location* button because the field on the grid is read only.

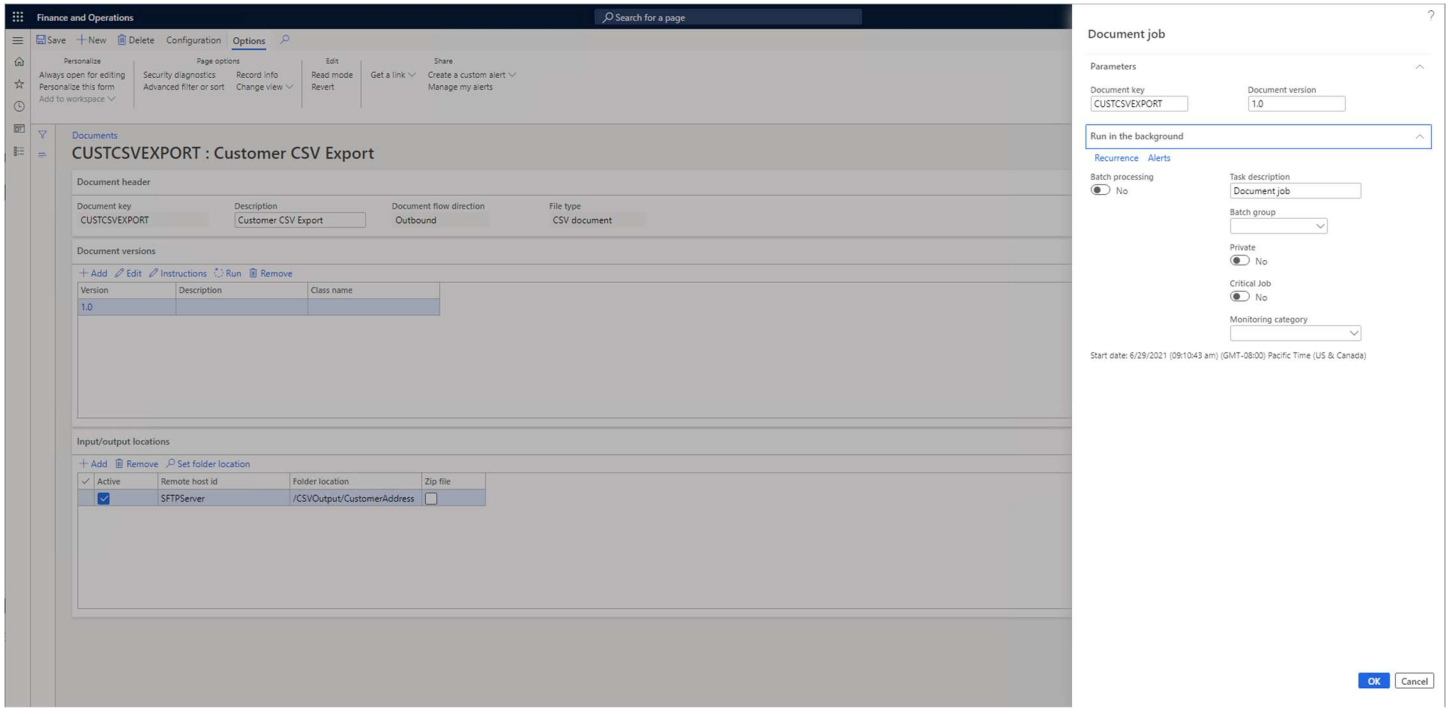
The *Set folder location* button will bring up a dialog like this one and you will select your destination folder from there:



Once you have selected the destination folder click on the *OK* button to exit the dialog.

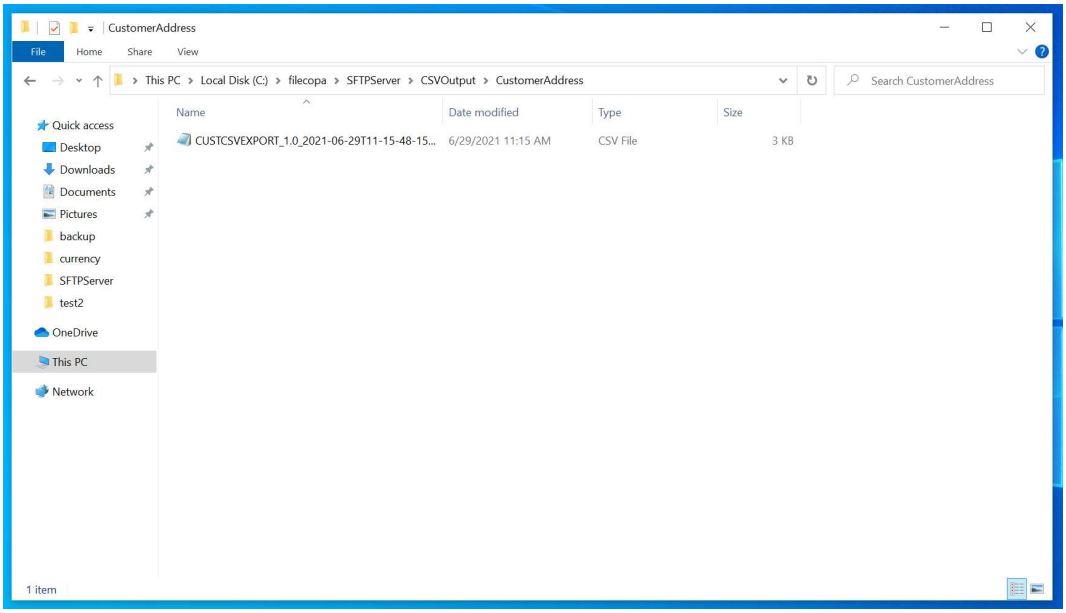


The Document and Document Version should now be configured correctly. Go to the *Document versions* grid, select version 1.0 and click on the *Run* button to execute version 1.0.



Make sure that *Batch processing* is set to *No*, click on the *OK* button.

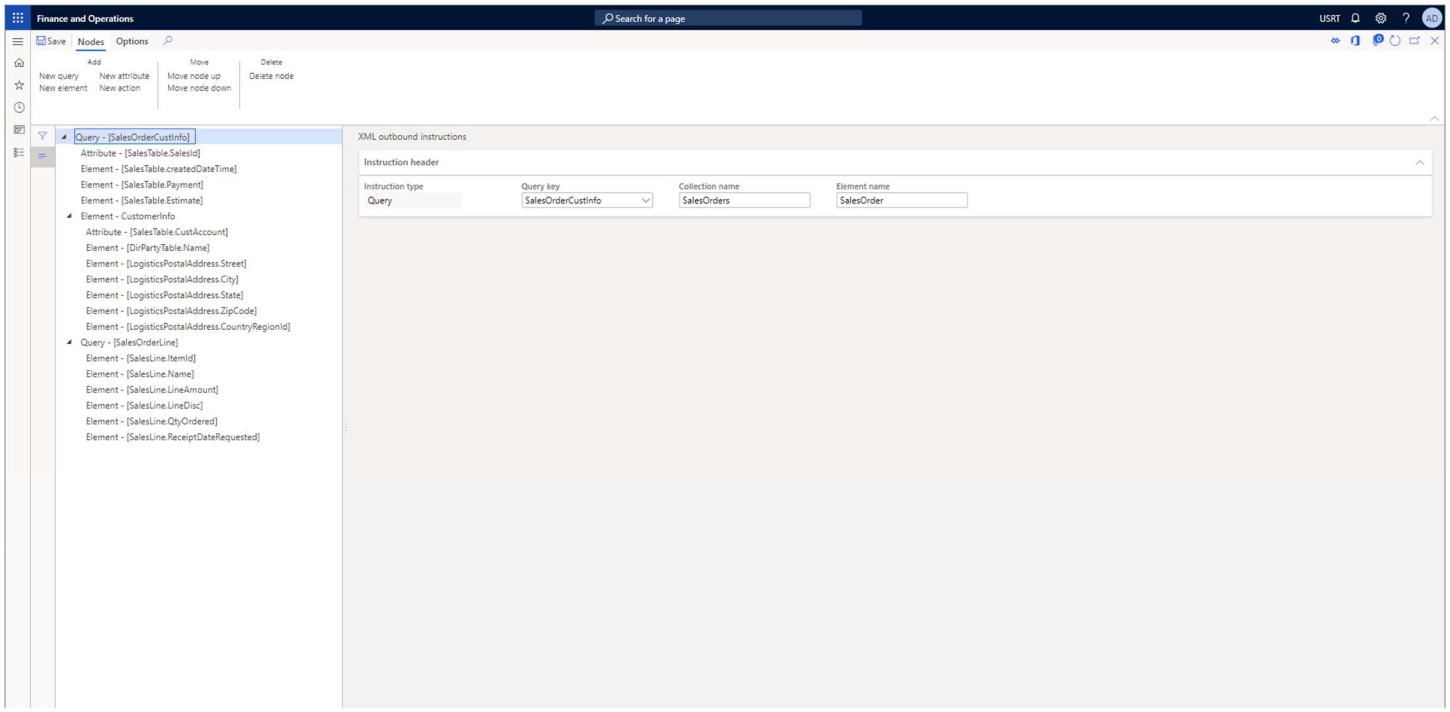
The system will generate a CSV file. In our case the file was placed in the */CSVOutput/CustomerAddress* folder in our *192.168.1.30* SFTP server.



Since we exported from the USRT legal entity our file has these contents:

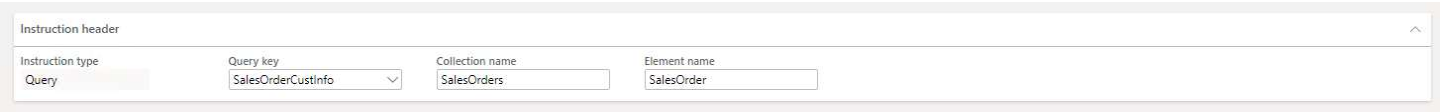
XML Document Export

The instructions for an outbound XML file are specified in a hierarchical tree style. The root element is a pointer to a Query object and the children can be Elements, Attributes, Actions, or other Queries. Queries can be parents of other objects.



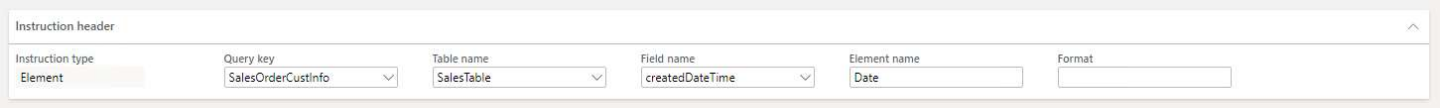
Instructions for an outbound XML document that creates the structure for a sales order. It has a header section, with a customer info section and a listing of sale lines.

A Query represents a table, or a set of tables as is defined by a Query.

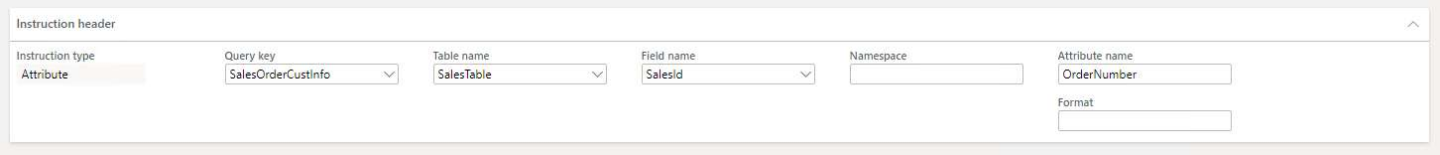


Child queries also have a Query Links section that determines how this Query item is linked to other ancestor Query items.

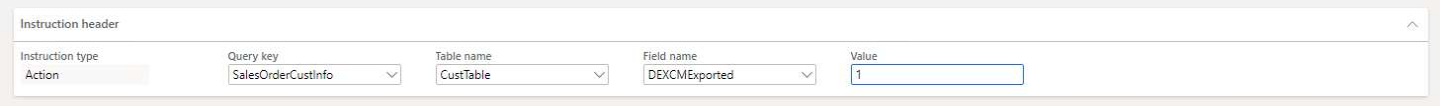
An Element item is the equivalent of a table/field pair from any parent Query expressed as an XML element tag.



An Attribute item is the equivalent of a table/field pair from any parent Query but expressed as an XML attribute tag.

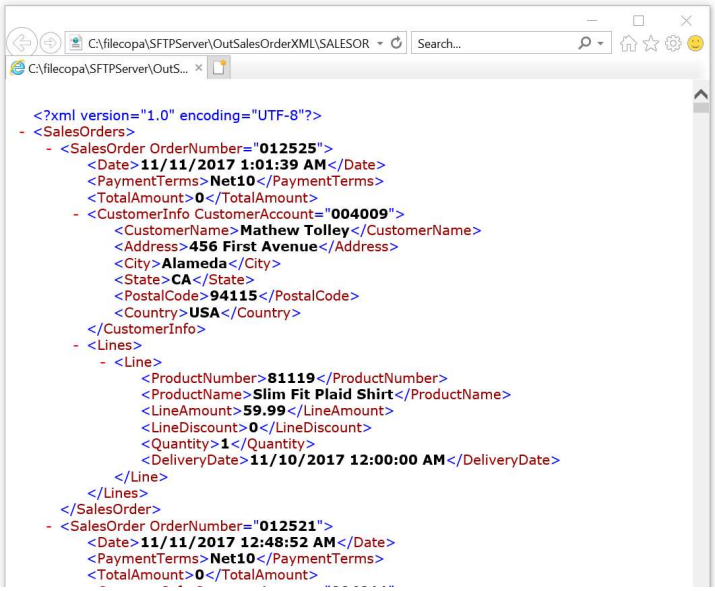


When Queries execute, Action items change values on table fields on the Queries. Action items are used to mark records that have been exported previously to allow incremental exports.



Sales Order XML Export Example

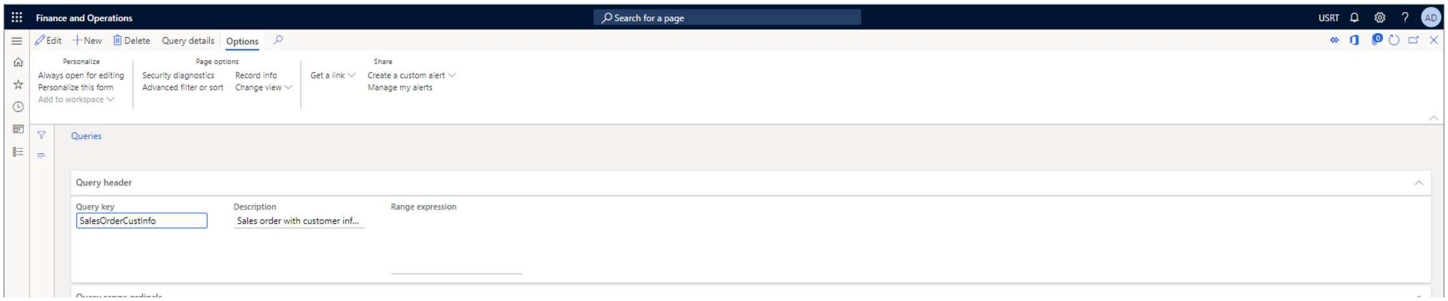
This section will walk you through the creation of an output XML file for a sales order like this one:



You will have to create two queries, a Query for the sales order header and the customer information and another Query for the sales lines.

To create a Query, go to: *Modules -> Document Exchange System -> Documents and queries -> Queries.*

Click on the *New* button.



Set the *Query key* control to *SalesOrderCustInfo*.

Set the *Description* control to *Sales order with customer information* (optional).

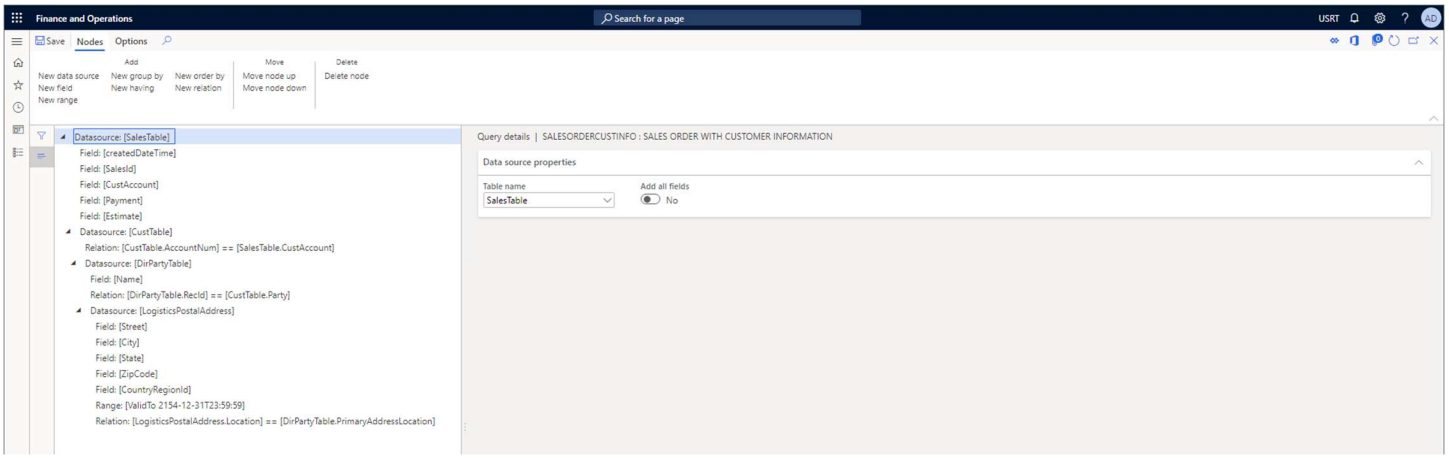
Click on the *Save* button.

Click on the *Query details* button.

Fill in the tree in the *Query details* form with these values:

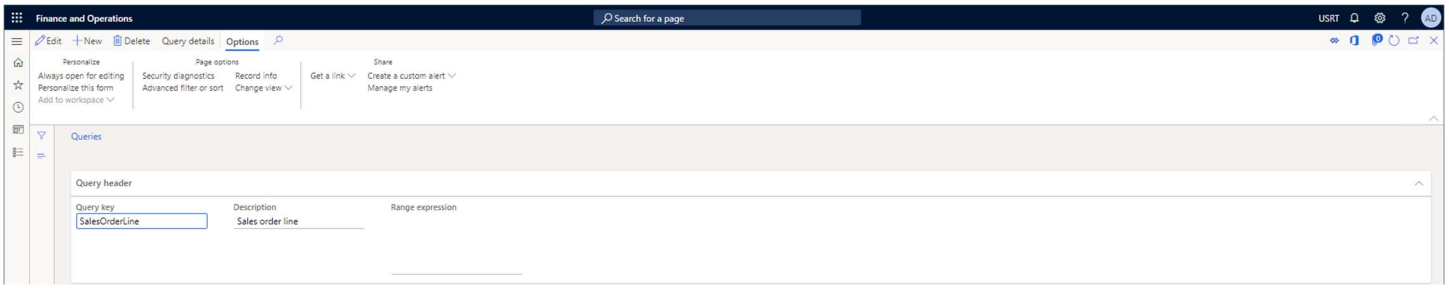
Datasource. *Table name:* SalesTable, *Add all fields*: No.
- **Field.** *Field name:* CreatedDateTime, *Query field type:* Standard.
- **Field.** *Field name:* SalesId, *Query field type:* Standard.
- **Field.** *Field name:* CustAccount, *Query field type:* Standard.
- **Field.** *Field name:* Payment, *Query field type:* Standard.
- **Field.** *Field name:* Estimate, *Query field type:* Standard.
- **Datasource.** *Table name:* CustTable, *Add all fields*: No.
  - **Relation.** *Field Name:* AccountNum, *Parent table name:* SalesTable, *Parent field name:* CustAccount.
  - **Datasource.** *Table name:* DirPartyTable, *Add all fields:* No.
    - **Field.** *Field name:* Name, *Query field type:* Standard.
    - **Relation.** *Field Name:* RecId, *Parent table name:* CustTable, *Parent field name:* Party.
    - **Datasource.** *Table name:* LogisticsPostalAddress, *Add all fields:* No.
      - **Field.** *Field name:* Street, *Query field type:* Standard.
      - **Field.** *Field name:* City, *Query field type:* Standard.
      - **Field.** *Field name:* State, *Query field type:* Standard.
      - **Field.** *Field name:* ZipCode, *Query field type:* Standard.
      - **Field.** *Field name:* CountryRegionId, *Query field type:* Standard.
      - **Range.** *Field name:* ValidTo, *Value:* 2154-12-31T23:59:59.
      - **Relation.** *Field Name:* Location, *Parent table name:* DirPartyTable, *Parent field name:* PrimaryAddressLocation.

After you have successfully entered all the above values your *Query details* tree should look like this:



To create second sales line Query again go to: *Modules -> Document Exchange System -> Documents and queries -> Queries.*

Click on the *New* button.



Set the *Query key* control to *SalesOrderLine*.

Set the *Description* control to *Sales order line* (optional).

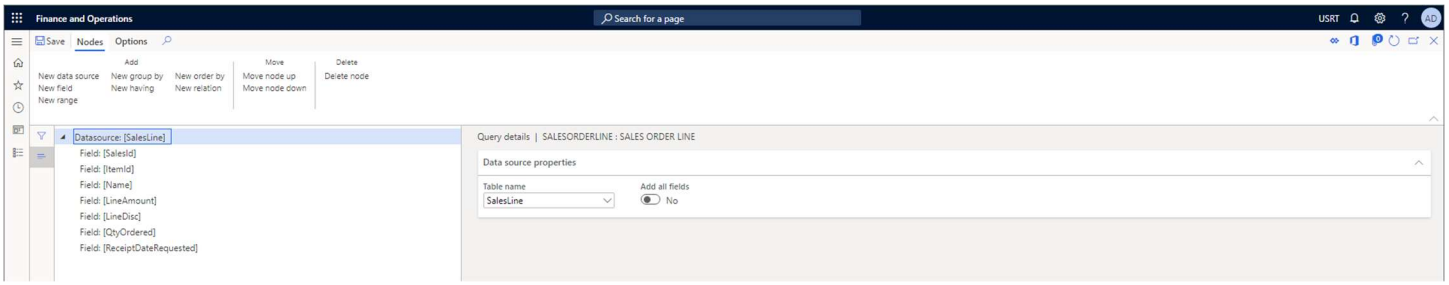
Click on the *Save* button.

Click on the *Query details* button.

Fill in the tree in the *Query details* form with these values:

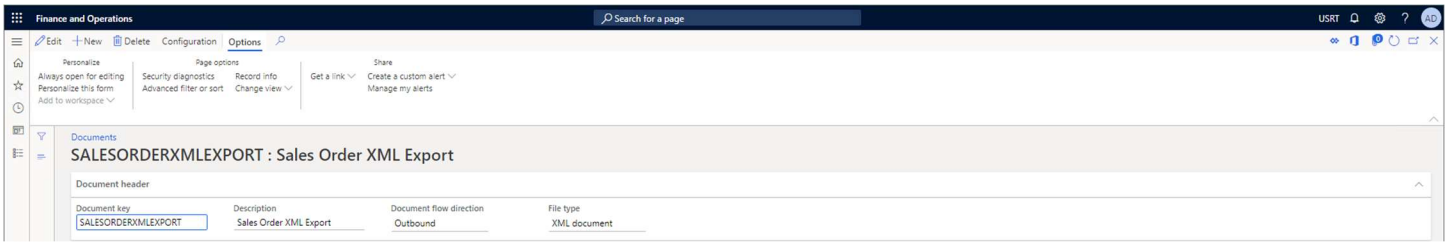
Datasource. *Table name:* SalesLine, *Add all fields*: No.
- **Field.** *Field name:* SalesId, *Query field type:* Standard.
- **Field.** *Field name:* ItemId, *Query field type:* Standard.
- **Field.** *Field name:* Name, *Query field type:* Standard.
- **Field.** *Field name:* LineAmount, *Query field type:* Standard.
- **Field.** *Field name:* LineDisc, *Query field type:* Standard.
- **Field.** *Field name:* QtyOrdered, *Query field type:* Standard.
- **Field.** *Field name:* ReceiptDateRequested, *Query field type:* Standard.

After you have successfully entered all the above values your *Query details* tree should look like this:



Now that both Queries are completed it is time to create a Document and a Document Version, go to: *Modules -> Document Exchange System -> Documents and queries -> Documents.*

Click on the *New* button.



Set the *Document Key* control to *SALESORDERXMLEXPORT*.

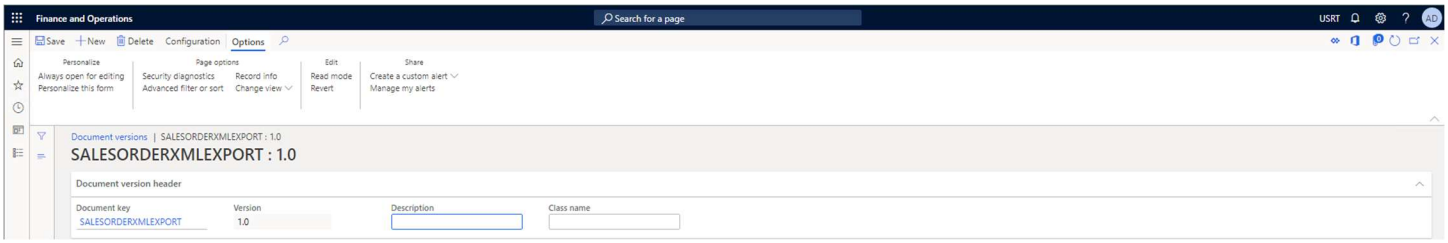
Set the *Description* control to *Sales Order XML Export* (optional).

Set the *Document flow direction* control to *Outbound*.

Set the *File type* control to *XML document*.

Click on the *Save* button.

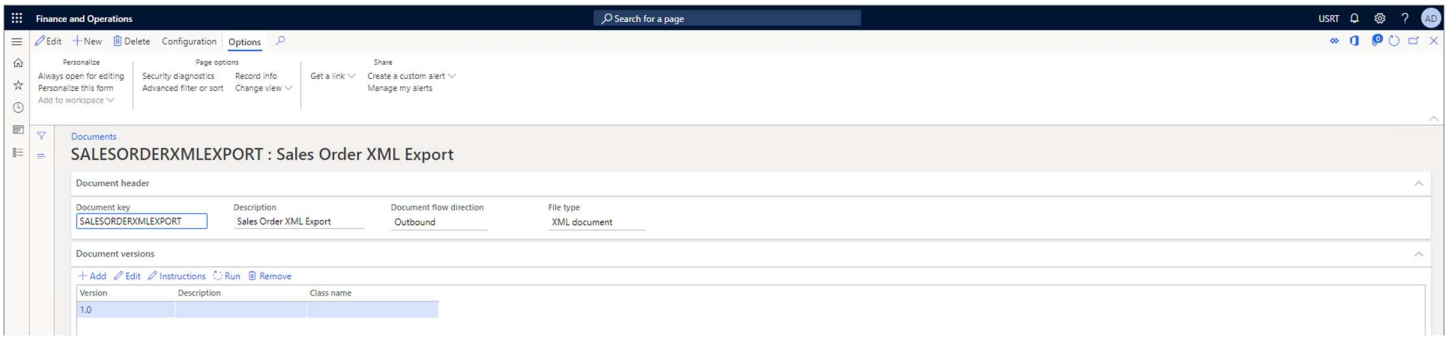
Click on the *Add* button in the *Document versions* grid to create a Document Version.



Set the *Version* control to *1.0*.

Exit the *Document versions* form.

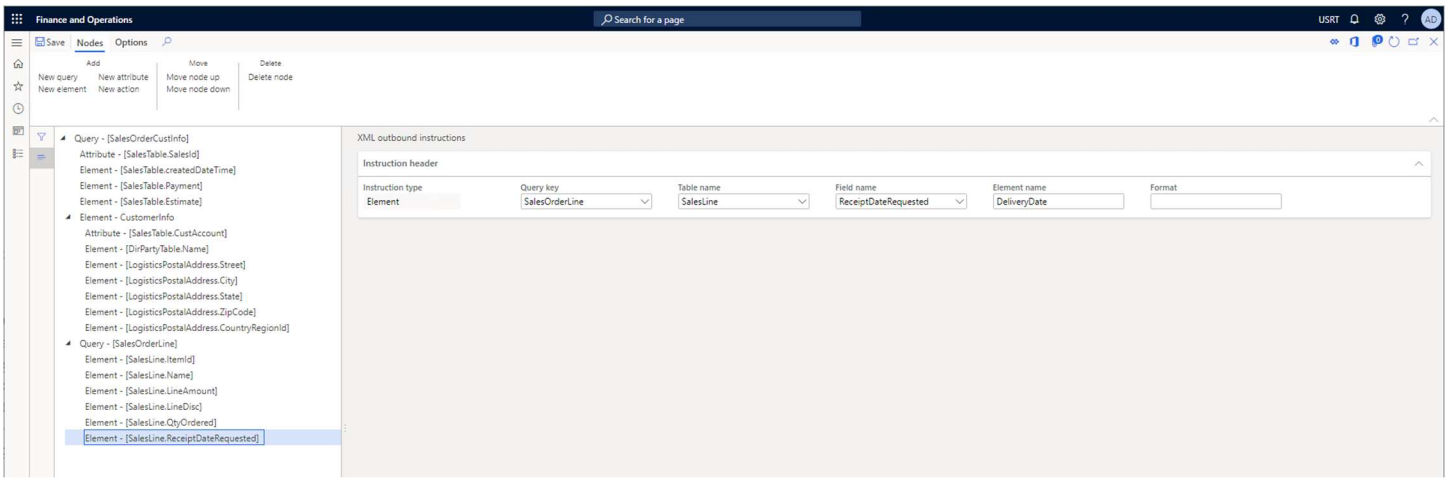
Click on the *Instructions* button in the *Document versions* grid.



Fill in the tree in the *XML outbound instructions* form with these values:

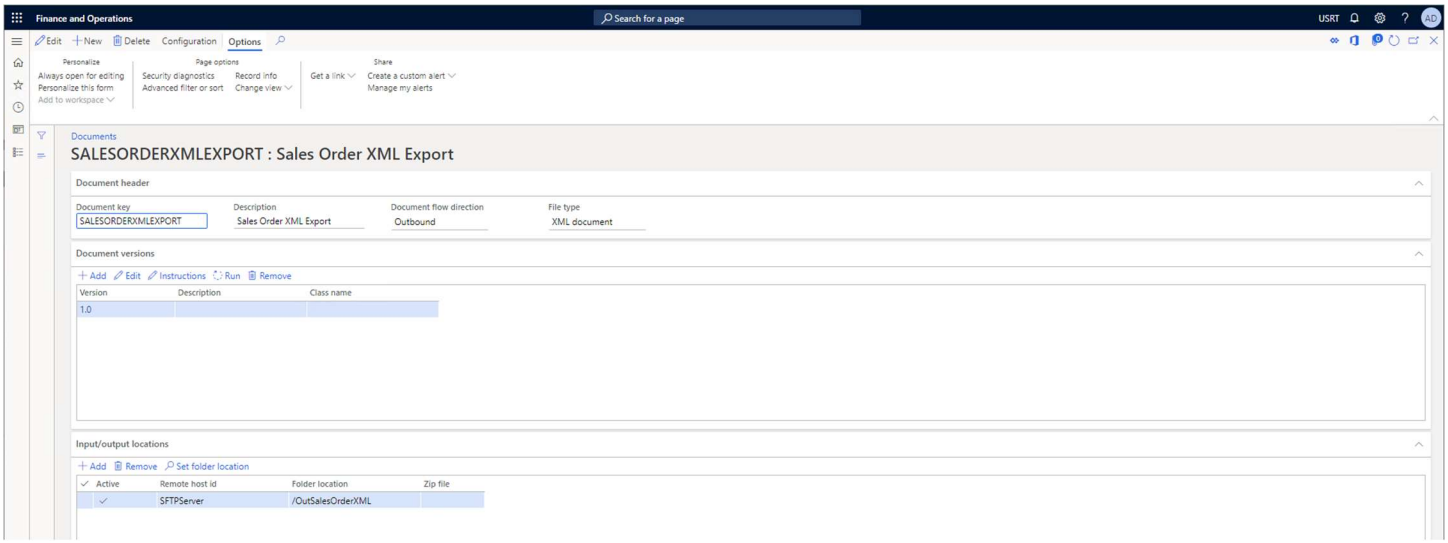
Query. *Query key:* SalesOrderCustInfo, *Collection name:* Sales Orders, *Element name:* Sales Order.
Attribute. *Query key:* SalesOrderCustInfo, *Table name:* SalesTable, *Field name:* SalesId, *Namespace:*, *Attribute name:* OrderNumber, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* SalesTable, *Field name:* CreatedDateTime, *Element name:* Date, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* SalesTable, *Field name:* Payment, *Element name:* PaymentTerms, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* SalesTable, *Field name:* Estimate, *Element name:* TotalAmount, *Format:*.
Element. *Query key:*, *Table name:*, *Field name:*, *Element name:* CustomerInfo, *Format:*.
Attribute. *Query key:* SalesOrderCustInfo, *Table name:* SalesTable, *Field name:* CustAccount, *Namespace:*, *Attribute name:* CustomerAccount, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* DirPartyTable, *Field name:* Name, *Element name:* CustomerName, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* LogisticsPostalAddress, *Field name:* Street, *Element name:* Address, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* LogisticsPostalAddress, *Field name:* City, *Element name:* City, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* LogisticsPostalAddress, *Field name:* State, *Element name:* State, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* LogisticsPostalAddress, *Field name:* ZipCode, *Element name:* PostalCode, *Format:*.
Element. *Query key:* SalesOrderCustInfo, *Table name:* LogisticsPostalAddress, *Field name:* CountryRegionId, *Element name:* Country, *Format:*.
Query. *Query key:* SalesOrderLine, *Collection name:* Lines, *Element name:* Line
Query Links. *Table name:* SalesLine, *Field name:* SalesId, *Query key:* SalesOrderCustInfo, *Parent table name:* SalesTable, *Parent field* name: SalesId.
Element. *Query key:* SalesOrderLine, *Table name:* SalesLine, *Field name:* ItemId, *Element name:* ProductNumber, *Format:*.
Element. *Query key:* SalesOrderLine, *Table name:* SalesLine, *Field name:* Name, *Element name:* ProductName, *Format:*.
Element. *Query key:* SalesOrderLine, *Table name:* SalesLine, *Field name:* LineAmount, *Element name:* LineAmount, *Format:*.
Element. *Query key:* SalesOrderLine, *Table name:* SalesLine, *Field name:* LineDisc, *Element name:* LineDiscount, *Format:*.
Element. *Query key:* SalesOrderLine, *Table name:* SalesLine, *Field name:* QtyOrdered, *Element name:* Quantity, *Format:*.
Element. *Query key:* SalesOrderLine, *Table name:* SalesLine, *Field name:* ReceiptDateRequested, *Element name:* DeliveryDate, *Format:*.

After you have successfully entered all the above values your *XML outbound instructions* tree should look like this:



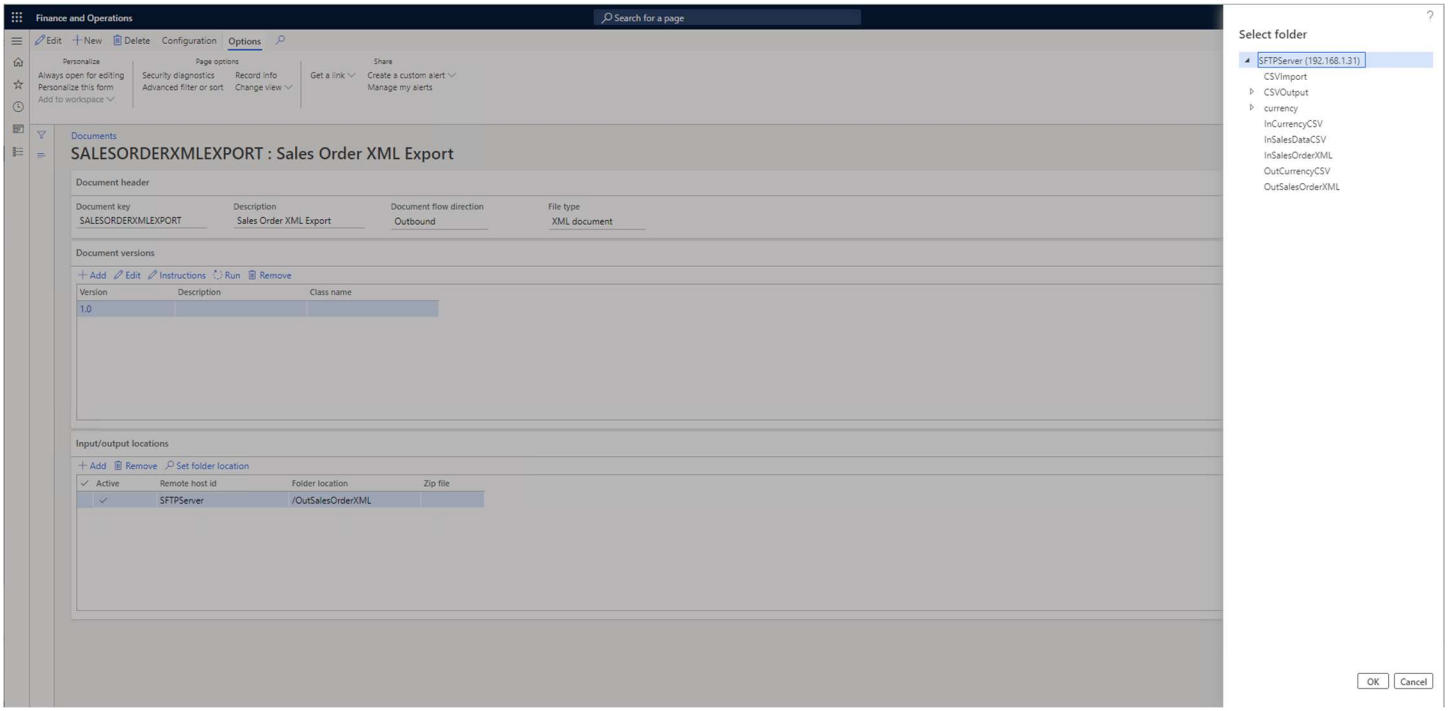
Exit the *XML outbound instructions* form.

In the *Input/output locations* grid you must add a Remote host, set that Remote host's *Folder location* and set that Remote host to *Active*. This will vary according to the setup on your system.



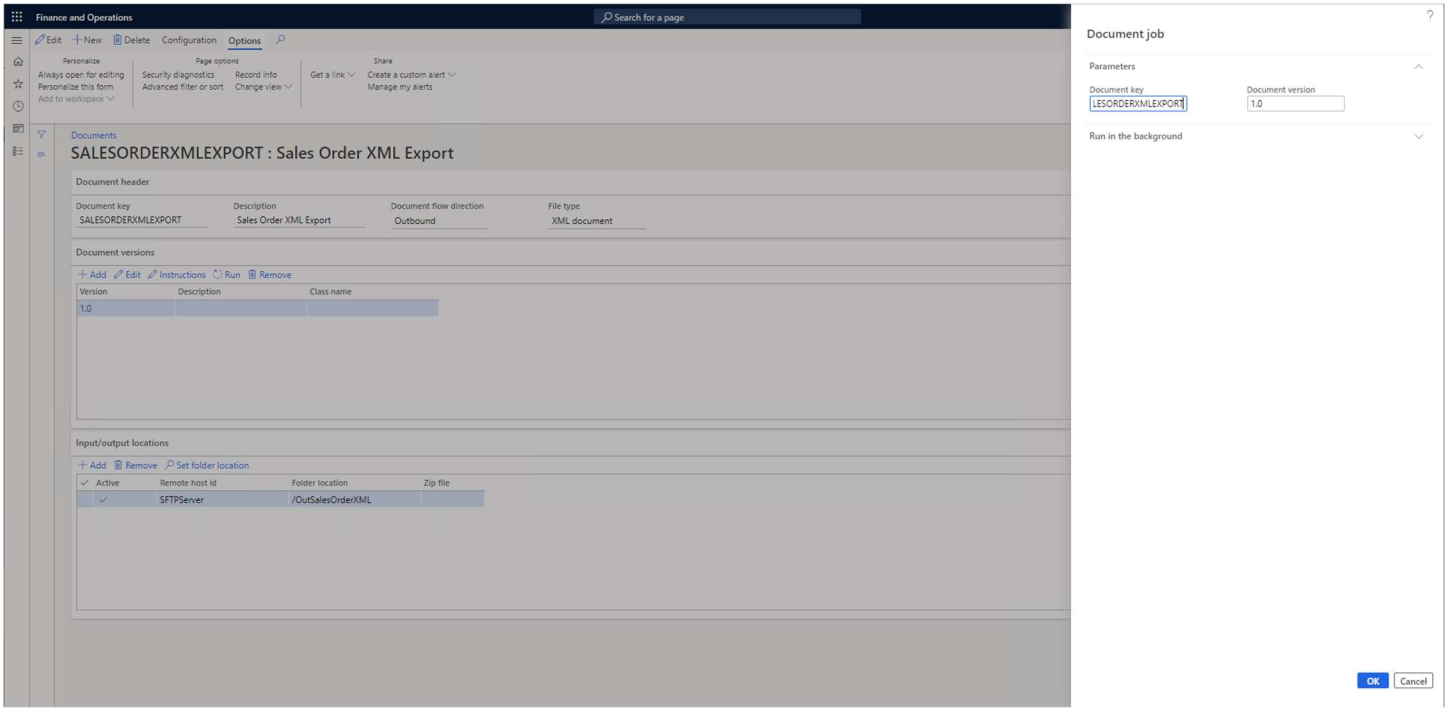
To set the *Folder location* use the *Set folder location* button because the field on the grid is read only.

The *Set folder location* button will bring up a dialog like this one and you will select your destination folder from there:



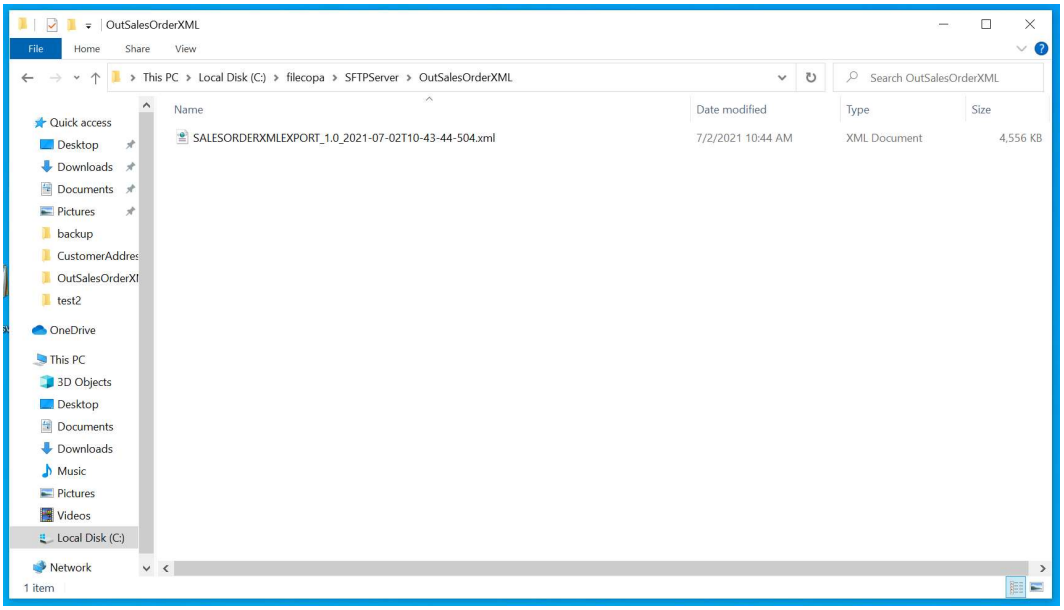
Once you have selected the destination folder click on the *OK* button to exit the dialog.

The Document and Document Version should now be configured correctly. Go to the *Document versions* grid, select version 1.0 and click on the *Run* button to execute version 1.0.

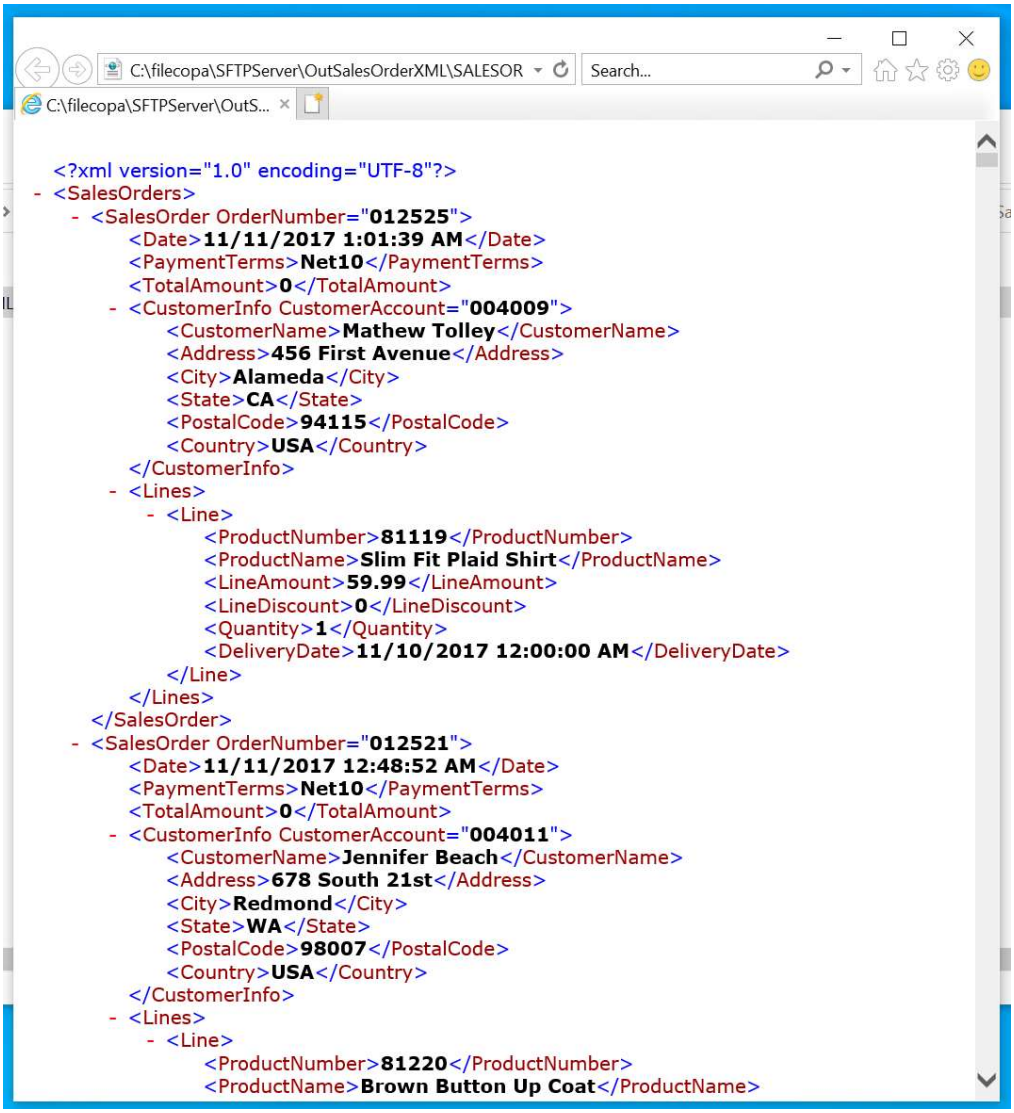


Make sure that *Batch processing* is set to *No*, click on the *OK* button.

The system will generate a XML file. In our case the file was placed in the */OutSalesOrder* folder in our *192.168.1.30* SFTP server.



Since we exported from the USRT legal entity our file has these contents:



Import System General Overview

Import a CSV Document.

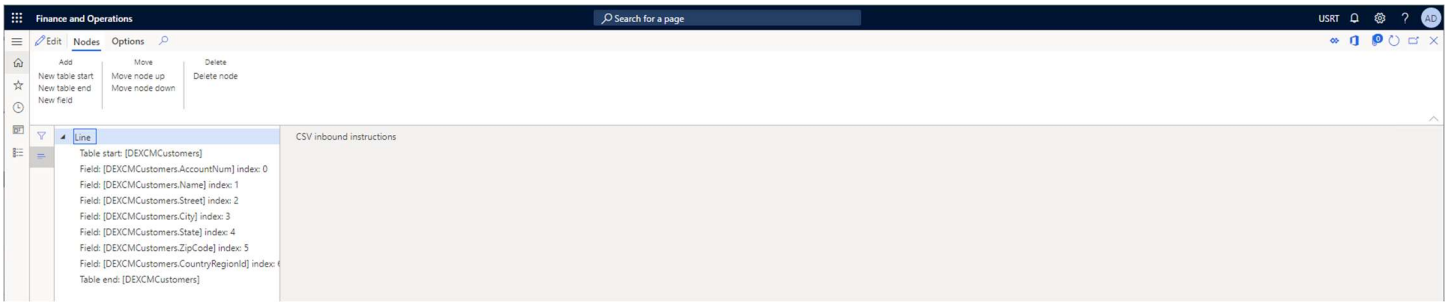
To import a CSV Document you create a Document and Document Version. You then create a set of Instructions that determine how a single line on that incoming CSV file maps to a Dynamics 365 table or tables. After the Instructions are defined you execute the Document Version. When you execute the Document Version, the CSV document is read from an SFTP, FTP, FTPS, Azure Storage or Azure Files server and then copied over to the temporary Azure Storage container and from there into the Dynamics 365 table or tables.

Import an XML Document.

To import an XML Document you create a Document and Document Version. You then create a set of Instructions that determine how an XML Document on that incoming XML file maps to a Dynamics 365 table or tables. After the Instructions are defined you execute the Document Version. When you execute the Document Version, the XML document is read from an SFTP, FTP, FTPS, Azure Storage or Azure Files server and then copied over to the temporary Azure Storage container and from there into the Dynamics 365 table or tables.

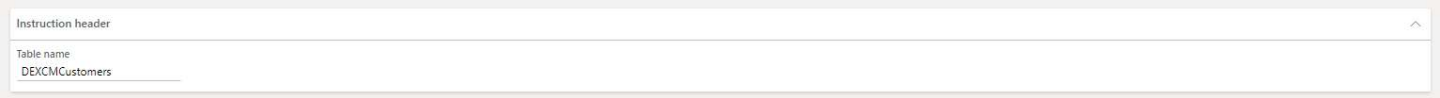
CSV Document Import

The instructions for an inbound CSV file are specified in a hierarchical tree style. The root item is a Line without any properties. The whole tree represents a single line on the input CSV file. Children items can be Table start, Table end and Field items.



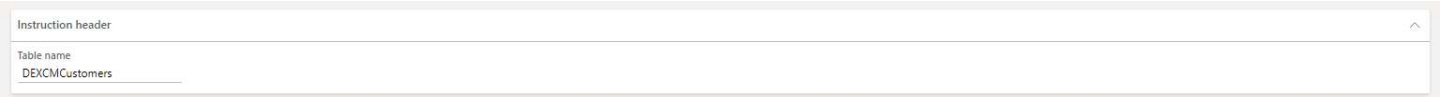
Instructions for an inbound CSV document. It will parse each line and separate fields by using the separator character. Fields in this line from the CSV file will be numbered starting with 0 for the first field. This tree also describes a staging table called DEXCMCustomers which will receive the data.

You can think of a Table start like instructing a staging table to create a new empty record.



In Field items you are copying data from the CSV file into the staging table's fields. When lines are parsed and separated into fields using the separator character, fields will be numbered starting with 0. The field number will be entered in the *Index* control. If the field in the incoming CSV file line is empty you can also set a text in the *Value* field that will replace it.

And you can think of a Table end like instructing the same staging table to insert the newly created record.



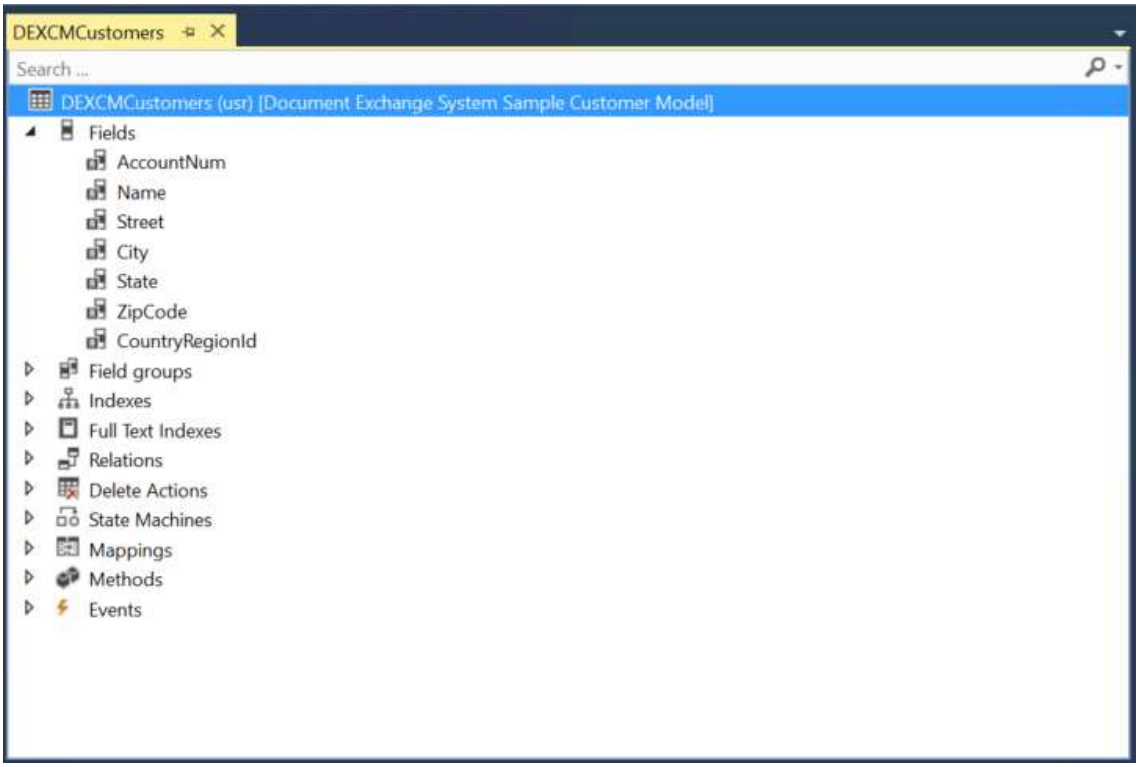
You can have as many Table start and Table end items as you need. The only requirement is that a Table Start must have a corresponding Table end.

Customer Address CSV Import Example

This section will walk you through the import of a CSV file with customer account, name, and address information like this one:

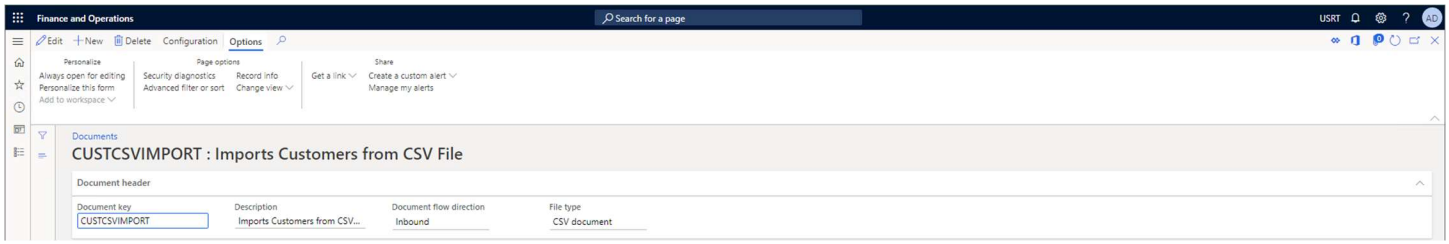
```
Account Number,Name,Address,City,State,Postal Code,Country
"004003","Mara Gentry","456 Ash Street","Oakland","CA","94115","USA"
"004005","Eve Whitehead","123 Oak Street","Redmond","WA","98007","USA"
"004007","Owen  Tolley","456 Sugar Hill","Tampa","FL","33601","USA"
"004009","Mathew Tolley","456 First Avenue","Alameda","CA","94115","USA"
"004011","Jennifer Beach","678 South 21st","Redmond","WA","98007","USA"
"004013","Shelly Beach","123 South Oak St","Renton","WA","98115","USA"
```

Into this D365 table:



Create a Document and a Document Version, go to: *Modules -> Document Exchange System -> Documents and queries -> Documents.*

Click on the *New* button.



Set the *Document Key* control to *CUSTCSVIMPORT*.

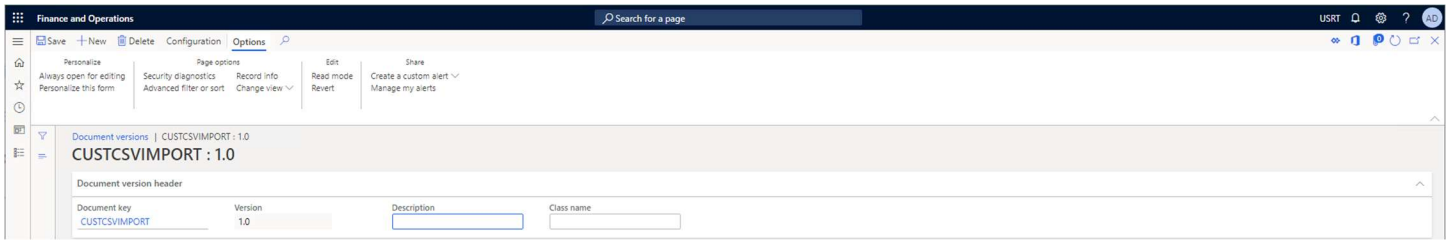
Set the *Description* control to *Imports Customers from CSV file* (optional).

Set the *Document flow direction* control to *Inbound*.

Set the *File type* control to *CSV Document*.

Click on the *Save* button.

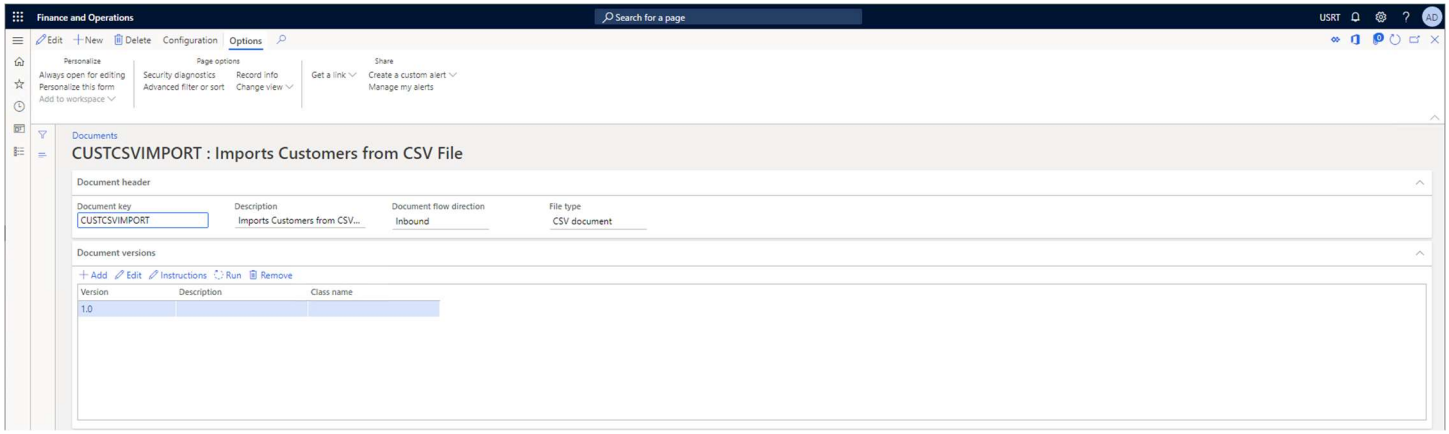
Click on the *Add* button in the *Document versions* grid to create a Document Version.



Set the *Version* control to *1.0*.

Exit the *Document versions* form.

Click on the *Instructions* button in the *Document versions* grid.

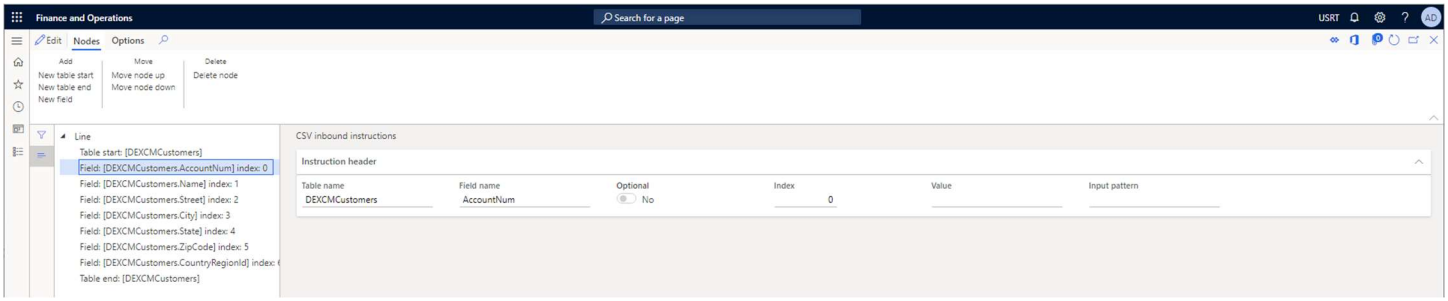


Fill in the tree in the *CSV inbound instructions* form with these values:

Line.

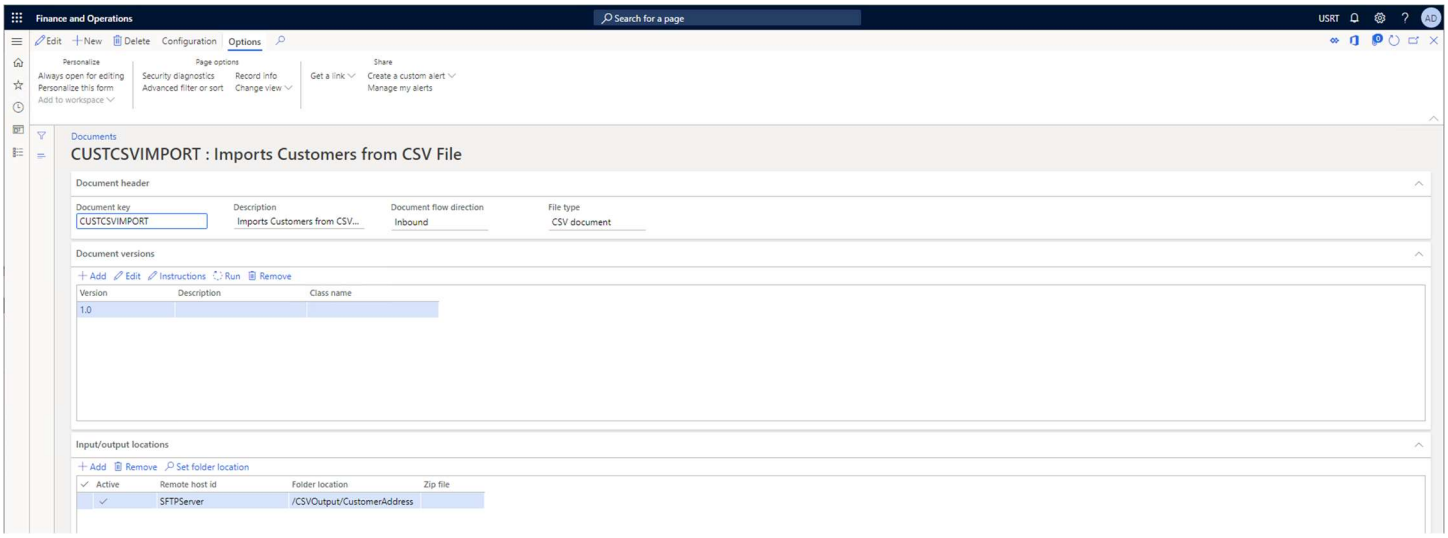
Table start. *Table name:* DEXCMCustomers.
Element. *Table name:* DEXCMCustomers, *Field name:* AccountNum, *Optional:* No, *Index*: 0, *Value:*, *Input pattern:*.
Element. *Table name:* DEXCMCustomers, *Field name:* Name, *Optional:* No, *Index*: 1, *Value:*, *Input pattern:*.
Element. *Table name:* DEXCMCustomers, *Field name:* Street, *Optional:* No, *Index*: 2, *Value:*, *Input pattern:*.
Element. *Table name:* DEXCMCustomers, *Field name:* City, *Optional:* No, *Index*: 3, *Value:*, *Input pattern:*.
Element. *Table name:* DEXCMCustomers, *Field name:* State, *Optional:* No, *Index*: 4, *Value:*, *Input pattern:*.
Element. *Table name:* DEXCMCustomers, *Field name:* ZipCode, *Optional:* No, *Index*: 5, *Value:*, *Input pattern:*.
Element. *Table name:* DEXCMCustomers, *Field name:* CountryRegionId, *Optional:* No, *Index*: 6, *Value:*, *Input pattern:*.
Table end. *Table name:* DEXCMCustomers.

After you have successfully entered all the above values your *CSV inbound instructions* tree should look like this:



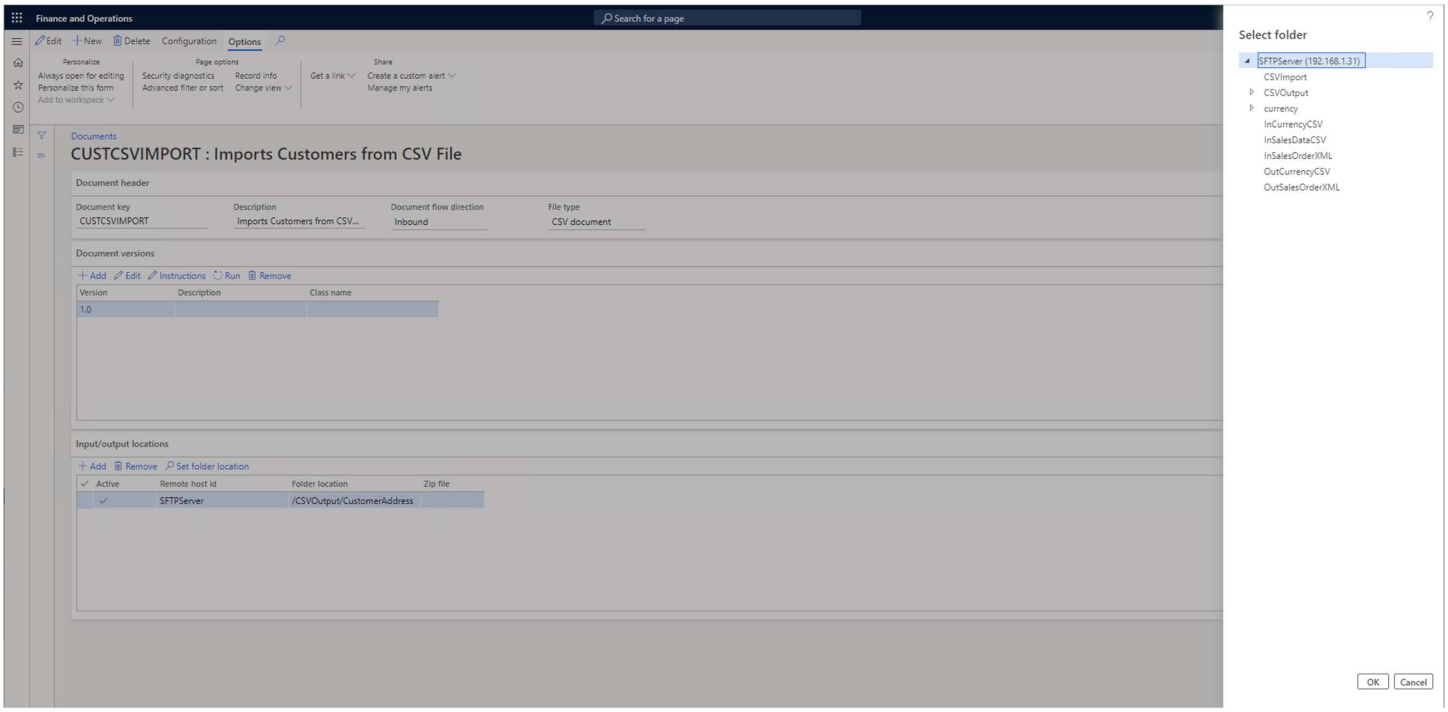
Exit the *CSV inbound instructions* form.

In the *Input/output locations* grid you must add a Remote host, set that Remote host's *Folder location* and set that Remote host to *Active*. This will vary according to the setup on your system.



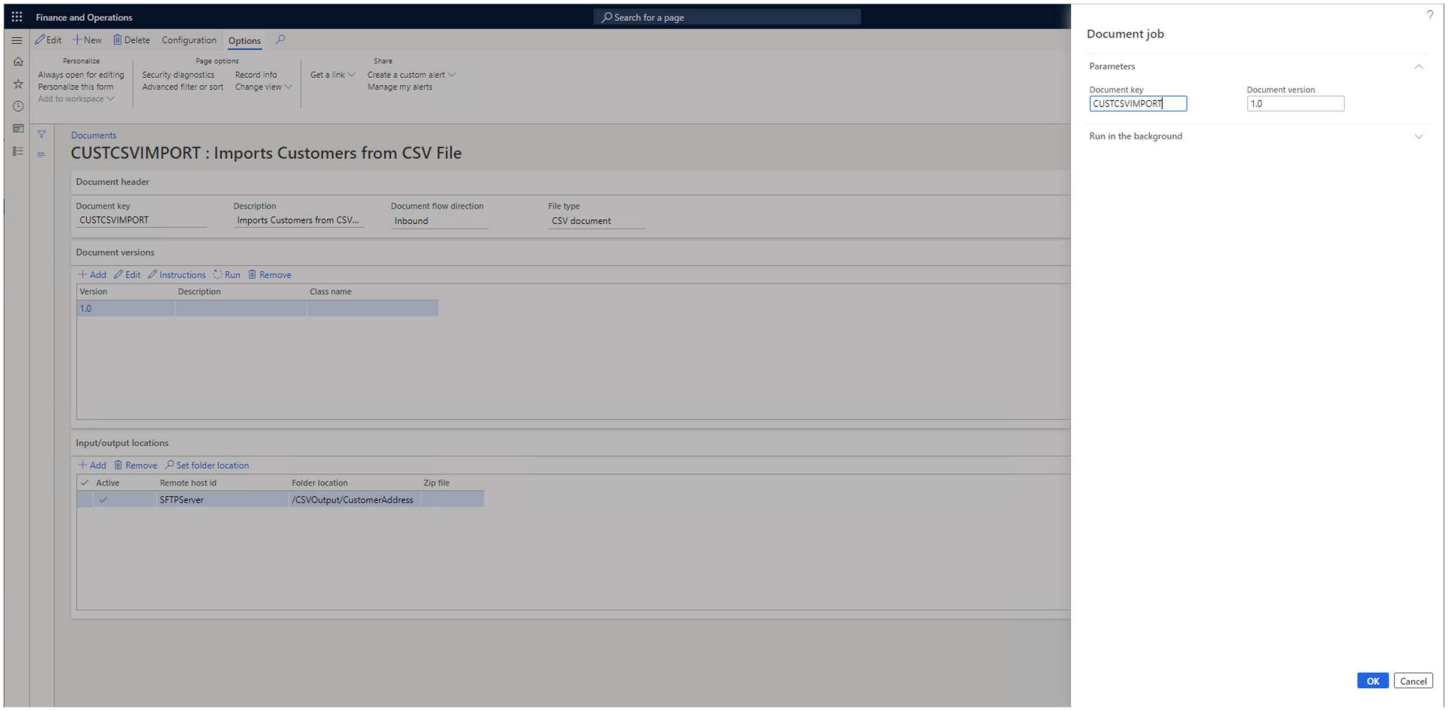
To set the *Folder location* use the *Set folder location* button because the field on the grid is read only.

The *Set folder location* button will bring up a dialog like this one and you will select your destination folder from there:



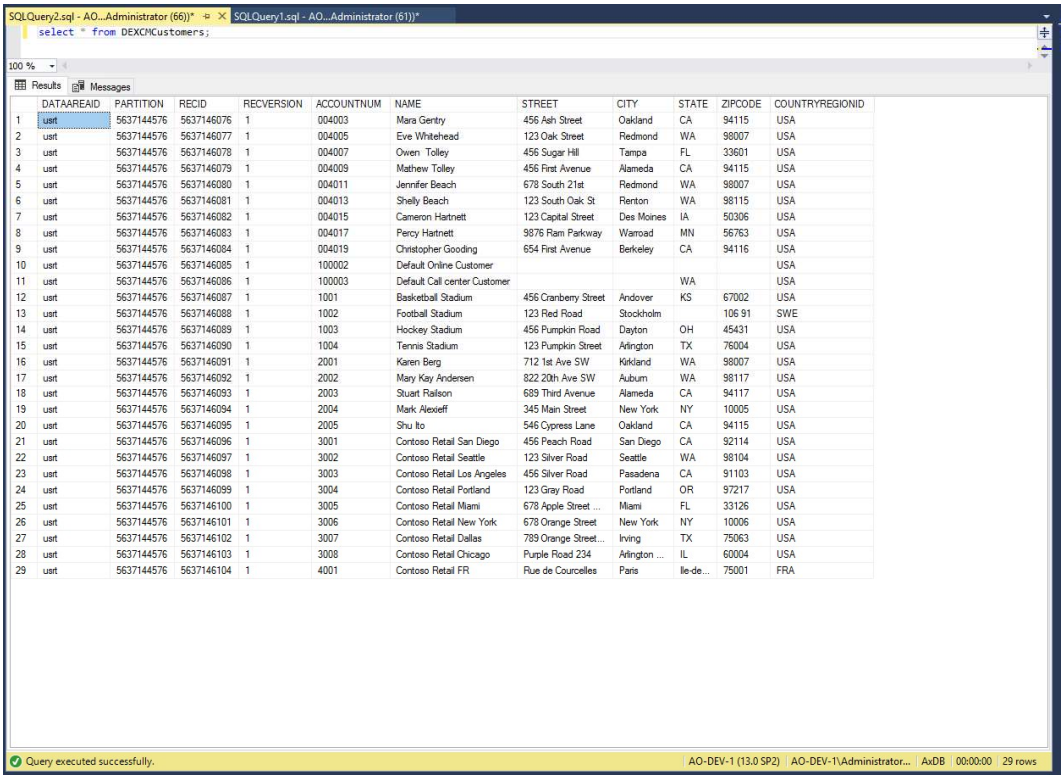
Once you have selected the destination folder click on the *OK* button to exit the dialog.

The Document and Document Version should now be configured correctly. Go to the *Document versions* grid, select version 1.0 and click on the *Run* button to execute version 1.0.



Make sure that *Batch processing* is set to *No*, click on the *OK* button.

The system will import the CSV file into the DEXCMCustomers D365 Table.



```
select * from DEXCMCustomers;
```

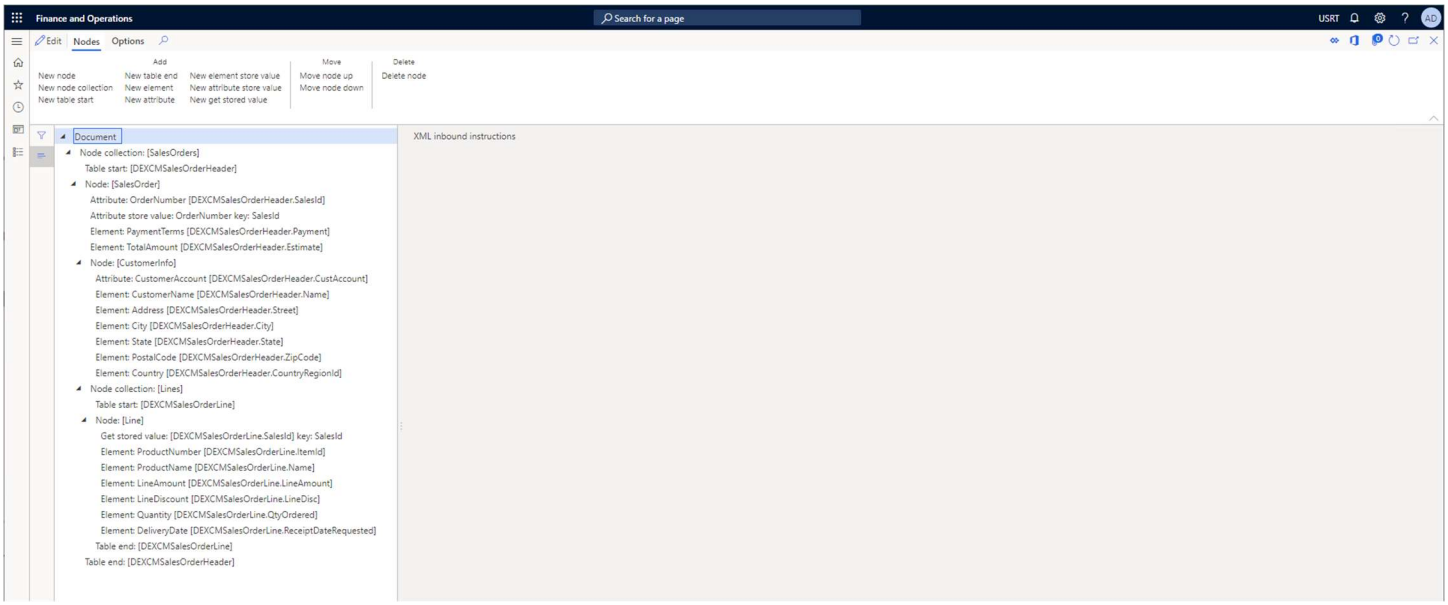
Results | Messages

	DATAAREAID	PARTITION	RECID	RECVERSION	ACCOUNTNUM	NAME	STREET	CITY	STATE	ZIPCODE	COUNTRYREGIONID
1	usrt	5637144576	5637146076	1	004003	Mara Gentry	456 Ash Street	Oakland	CA	94115	USA
2	usrt	5637144576	5637146077	1	004005	Eve Whitehead	123 Oak Street	Redmond	WA	98007	USA
3	usrt	5637144576	5637146078	1	004007	Owen Tolley	456 Sugar Hill	Tampa	FL	33601	USA
4	usrt	5637144576	5637146079	1	004009	Mathew Tolley	456 First Avenue	Alameda	CA	94115	USA
5	usrt	5637144576	5637146080	1	004011	Jennifer Beach	678 South 21st	Redmond	WA	98007	USA
6	usrt	5637144576	5637146081	1	004013	Shelly Beach	123 South Oak St	Renton	WA	98115	USA
7	usrt	5637144576	5637146082	1	004015	Cameron Hartnett	123 Capital Street	Des Moines	IA	50306	USA
8	usrt	5637144576	5637146083	1	004017	Percy Hartnett	9876 Ram Parkway	Warroad	MN	56763	USA
9	usrt	5637144576	5637146084	1	004019	Christopher Gooding	654 First Avenue	Berkeley	CA	94116	USA
10	usrt	5637144576	5637146085	1	100002	Default Online Customer					USA
11	usrt	5637144576	5637146086	1	100003	Default Call center Customer			WA		USA
12	usrt	5637144576	5637146087	1	1001	Basketball Stadium	456 Cranberry Street	Andover	KS	67002	USA
13	usrt	5637144576	5637146088	1	1002	Football Stadium	123 Red Road	Stockholm		106 91	SWE
14	usrt	5637144576	5637146089	1	1003	Hockey Stadium	456 Pumpkin Road	Dayton	OH	45431	USA
15	usrt	5637144576	5637146090	1	1004	Tennis Stadium	123 Pumpkin Street	Arlington	TX	76004	USA
16	usrt	5637144576	5637146091	1	2001	Karen Berg	712 1st Ave SW	Kirkland	WA	98007	USA
17	usrt	5637144576	5637146092	1	2002	Mary Kay Andersen	822 20th Ave SW	Auburn	WA	98117	USA
18	usrt	5637144576	5637146093	1	2003	Stuart Railson	689 Third Avenue	Alameda	CA	94117	USA
19	usrt	5637144576	5637146094	1	2004	Mark Alexieff	345 Main Street	New York	NY	10005	USA
20	usrt	5637144576	5637146095	1	2005	Shu Ito	546 Cypress Lane	Oakland	CA	94115	USA
21	usrt	5637144576	5637146096	1	3001	Contoso Retail San Diego	456 Peach Road	San Diego	CA	92114	USA
22	usrt	5637144576	5637146097	1	3002	Contoso Retail Seattle	123 Silver Road	Seattle	WA	98104	USA
23	usrt	5637144576	5637146098	1	3003	Contoso Retail Los Angeles	456 Silver Road	Pasadena	CA	91103	USA
24	usrt	5637144576	5637146099	1	3004	Contoso Retail Portland	123 Gray Road	Portland	OR	97217	USA
25	usrt	5637144576	5637146100	1	3005	Contoso Retail Miami	678 Apple Street ...	Miami	FL	33126	USA
26	usrt	5637144576	5637146101	1	3006	Contoso Retail New York	678 Orange Street	New York	NY	10006	USA
27	usrt	5637144576	5637146102	1	3007	Contoso Retail Dallas	789 Orange Street...	Irving	TX	75063	USA
28	usrt	5637144576	5637146103	1	3008	Contoso Retail Chicago	Purple Road 234	Arlington ...	IL	60004	USA
29	usrt	5637144576	5637146104	1	4001	Contoso Retail FR	Rue de Courcelles	Paris	Ile-de...	75001	FRA

Query executed successfully. AO-DEV-1 (13.0 SP2) | AO-DEV-1\Administrator... | AxDB | 00:00:00 | 29 rows

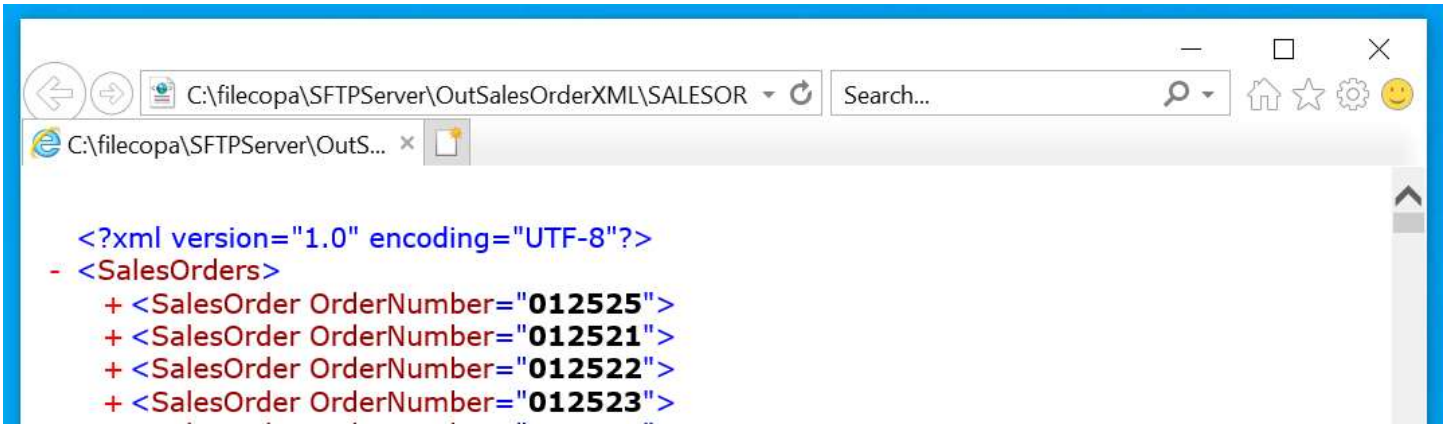
XML Document Import

The instructions for an inbound CSV file are specified in a hierarchical tree style. The root item is a Document without any properties. The whole tree is a representation of an XML document in an incoming file. A document can have Node, Node Collection, Element and Attribute items plus Table start and Table end items. The Node, Node Collection, Element and Attribute items are used to map the structure of the XML document. The Table start and Table end items are there to signal the creation and insertion of records into Dynamics 365 tables. Element store value, Attribute store value and Get stored value items are there to save and then retrieve values from the XML document, because the structure of the XML document might not always agree with Dynamics 365 tables.



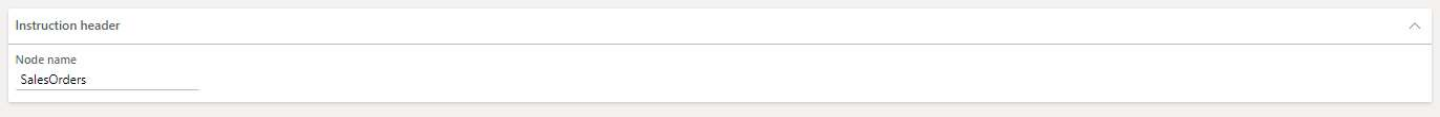
```
Document
  Node collection: [SalesOrders]
    Table start: [DEXCMSalesOrderHeader]
    Node: [SalesOrder]
      Attribute: OrderNumber [DEXCMSalesOrderHeader.SalesId]
      Attribute store value: OrderNumber key: SalesId
      Element: PaymentTerms [DEXCMSalesOrderHeader.Payment]
      Element: TotalAmount [DEXCMSalesOrderHeader.Estimate]
      Node: [CustomerInfo]
        Attribute: CustomerAccount [DEXCMSalesOrderHeader.CustAccount]
        Element: CustomerName [DEXCMSalesOrderHeader.Name]
        Element: Address [DEXCMSalesOrderHeader.Street]
        Element: City [DEXCMSalesOrderHeader.City]
        Element: State [DEXCMSalesOrderHeader.State]
        Element: PostalCode [DEXCMSalesOrderHeader.ZipCode]
        Element: Country [DEXCMSalesOrderHeader.CountryRegionId]
      Node collection: [Lines]
        Table start: [DEXCMSalesOrderLine]
        Node: [Line]
          Get stored value: [DEXCMSalesOrderLine.SalesId] key: SalesId
          Element: ProductNumber [DEXCMSalesOrderLine.ItemId]
          Element: ProductName [DEXCMSalesOrderLine.Name]
          Element: LineAmount [DEXCMSalesOrderLine.LineAmount]
          Element: LineDiscount [DEXCMSalesOrderLine.LineDisc]
          Element: Quantity [DEXCMSalesOrderLine.QtyOrdered]
          Element: DeliveryDate [DEXCMSalesOrderLine.ReceiptDateRequested]
        Table end: [DEXCMSalesOrderLine]
    Table end: [DEXCMSalesOrderHeader]
```

A Node collection item maps this type of element in an XML file:



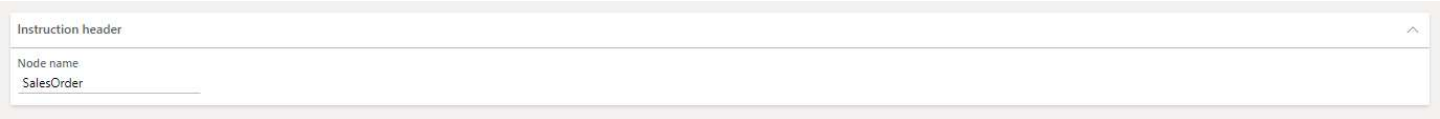
```
  <?xml version="1.0" encoding="UTF-8"?>
- <SalesOrders>
   + <SalesOrder OrderNumber="012525">
   + <SalesOrder OrderNumber="012521">
   + <SalesOrder OrderNumber="012522">
   + <SalesOrder OrderNumber="012523">
```

Here we have a collection element called SalesOrders and all the child items are all SalesOrder elements.



In a standard Node all the child items are of different types:

```
<SalesOrder OrderNumber="012525">
    <Date>11/11/2017 1:01:39 AM</Date>
    <PaymentTerms>Net10</PaymentTerms>
    <TotalAmount>0</TotalAmount>
  + <CustomerInfo CustomerAccount="004009">
  + <Lines>
</SalesOrder>
```

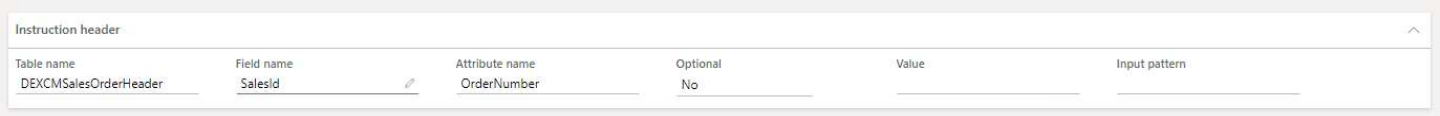


An Element does not have any children and can have a value or be empty:

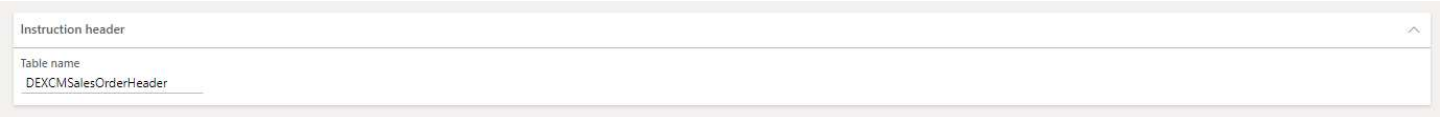
```
<PaymentTerms>Net30</PaymentTerms>
```

An Attribute is a property on an Element or Node:

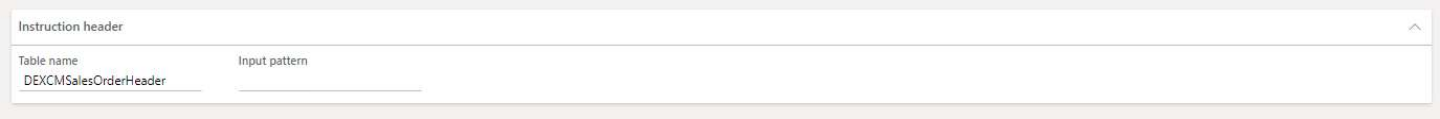
```
<SalesOrder OrderNumber="012525">
```



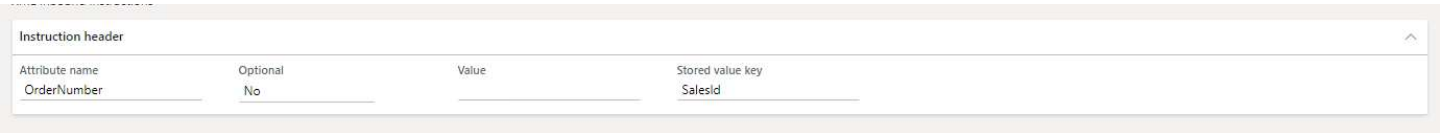
A Table start signals the creation of a new record on a D365 table.



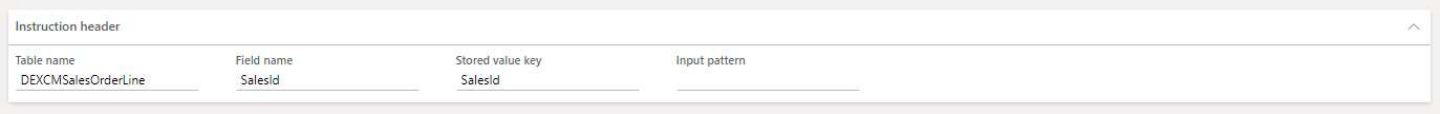
A Table end item signals the insert operation on a D365 table.



An Element or Attribute store value item retrieves a value for an element/attribute and keeps it in memory for future use.

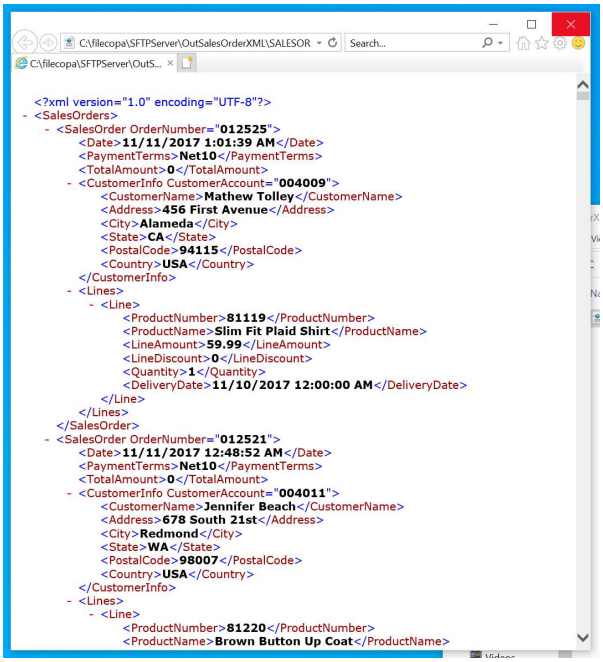


A Get stored value item retrieves a value stored by an Element or Attribute store value and places that value on a D365 table.

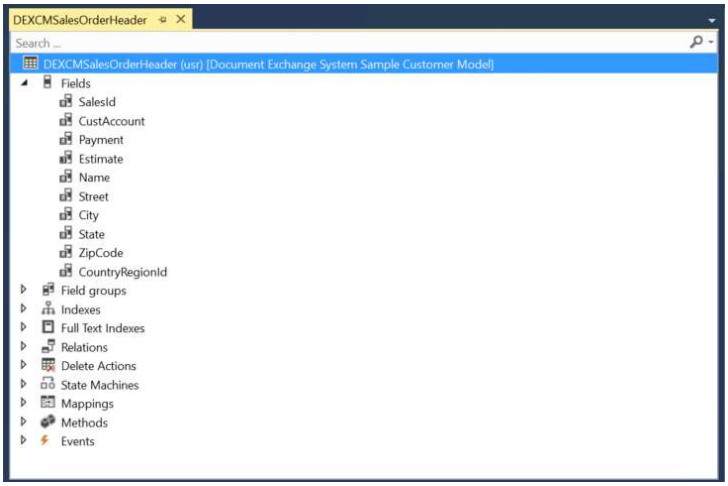


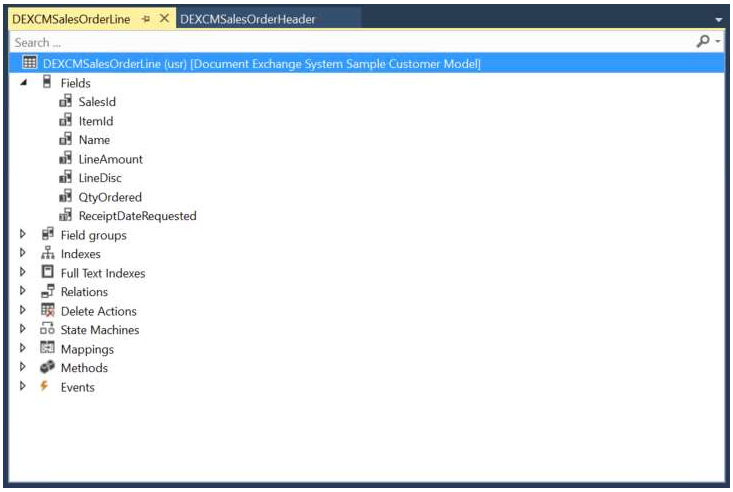
Sales Order Import Example

This section will walk you through the import of a XML file with sales order information like this one:



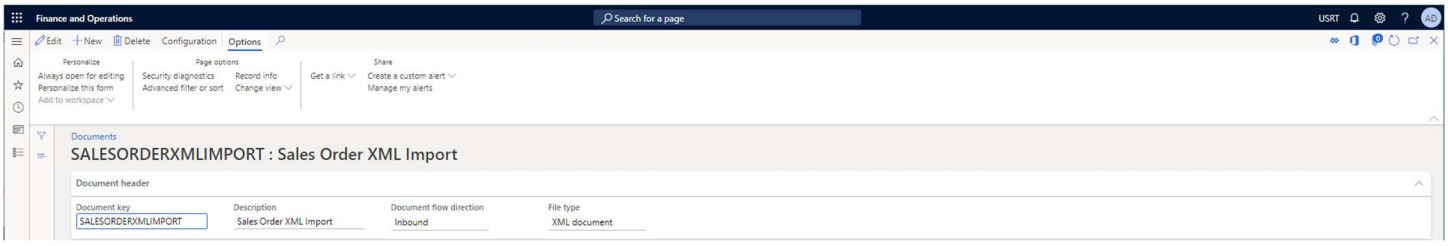
Into these two D365 tables:





Create a Document and a Document Version, go to: *Modules -> Document Exchange System -> Documents and queries -> Documents.*

Click on the *New* button.



Set the *Document Key* control to *SALESORDERXMLImport*.

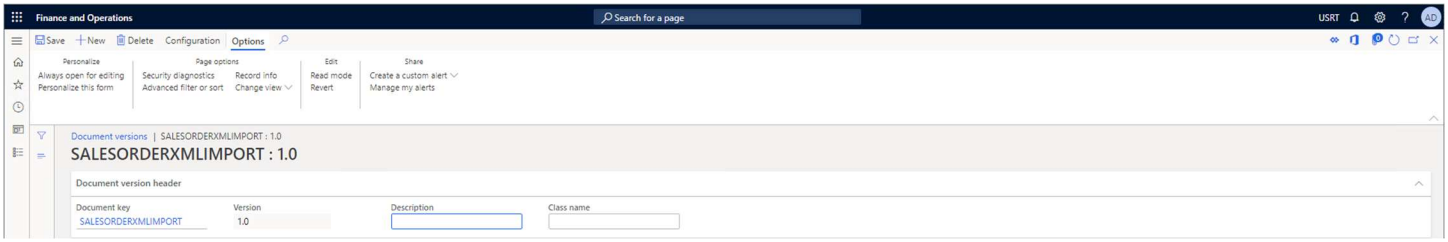
Set the *Description* control to *Sales Order XML Import* (optional).

Set the *Document flow direction* control to *Inbound*.

Set the *File type* control to *XML Document*.

Click on the *Save* button.

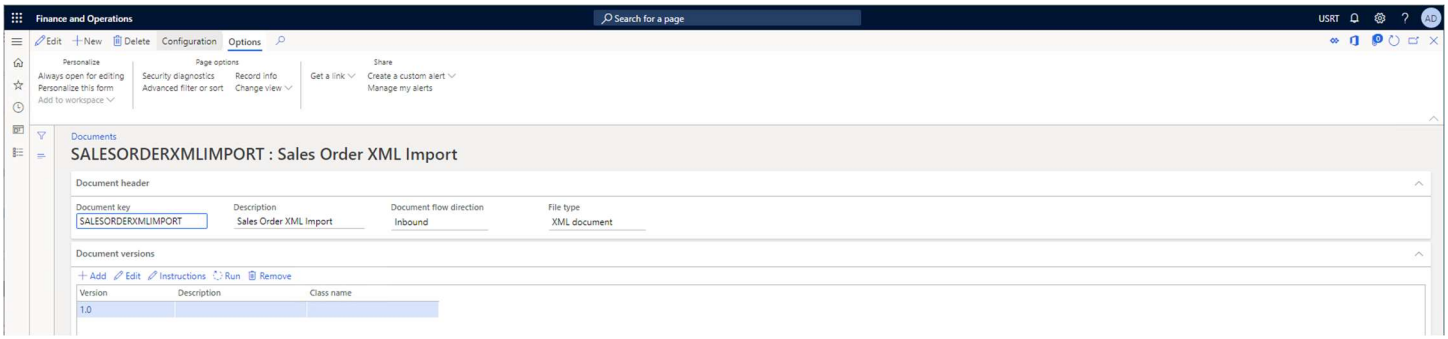
Click on the *Add* button in the *Document versions* grid to create a Document Version.



Set the *Version* control to *1.0*.

Exit the *Document versions* form.

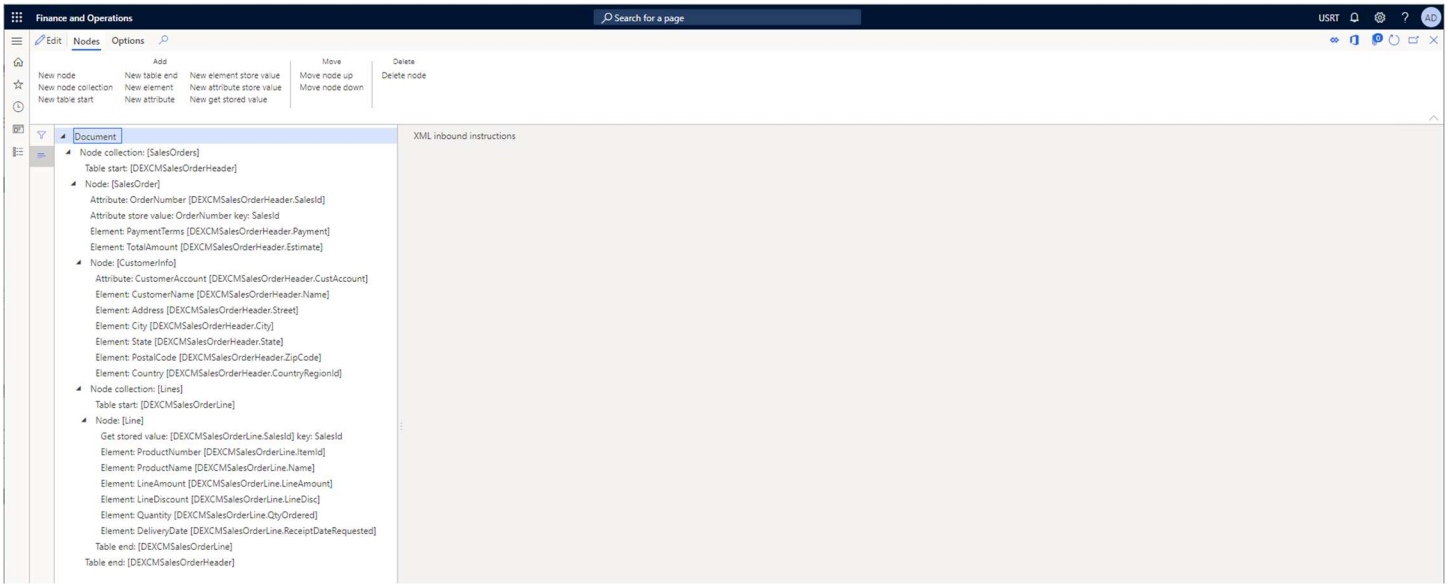
Click on the *Instructions* button in the *Document versions* grid.



Fill in the tree in the *XML inbound instructions* form with these values:

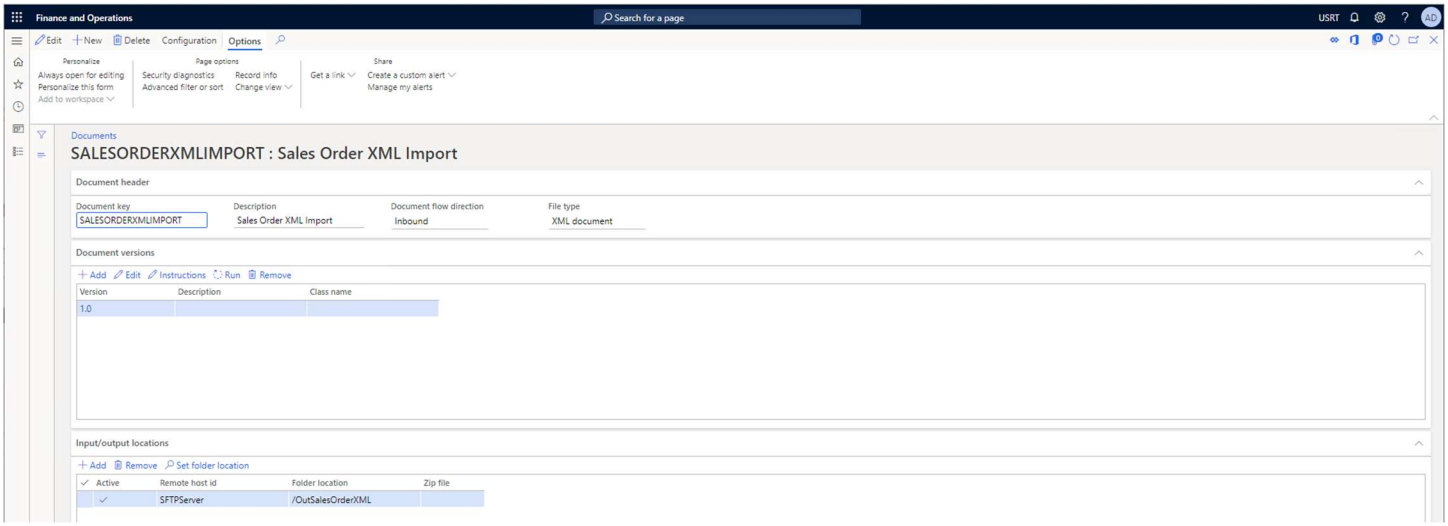
Document.
- **Node collection.** *Node name:* SalesOrders.
  - **Table start.** *Table name:* DEXCMSalesOrderHeader.
  - **Node.** *Node name:* SalesOrder.
    - **Attribute.** *Table name:* DEXCMSalesOrderHeader, *Field name:* SalesId, *Attribute name:* OrderNumber, *Optional:* No, *Value:*, *Input pattern:*.
    - **Attribute store value.** *Attribute name:* OrderNumber, *Optional:* No, *Value:*, *Stored value key:* SalesId.
    - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* Payment, *Element name:* PaymentTerms, *Optional:* No, *Value:*, *Input pattern:*.
    - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* Estimate, *Element name:* TotalAmount, *Optional:* No, *Value:*, *Input pattern:*.
    - **Node.** *Node name:* CustomerInfo.
      - **Attribute.** *Table name:* DEXCMCustomers.
      - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* Name, *Element name:* CustomerName, *Optional:* No, *Value:*, *Input pattern:*.
      - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* Street, *Element name:* Address, *Optional:* No, *Value:*, *Input pattern:*.
      - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* City, *Element name:* City, *Optional:* No, *Value:*, *Input pattern:*.
      - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* State, *Element name:* State, *Optional:* No, *Value:*, *Input pattern:*.
      - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* ZipCode, *Element name:* PostalCode, *Optional:* No, *Value:*, *Input pattern:*.
      - **Element.** *Table name:* DEXCMSalesOrderHeader, *Field name:* CountryRegionId, *Element name:* Country, *Optional:* No, *Value:*, *Input pattern:*.
    - **Node collection.** *Node name:* Lines.
      - **Table start.** *Table name:* DEXCMSalesOrderLine.
      - **Node.** *Node name:* Line.
        - **Get stored value.** *Table name:* DEXCMCustomers, *Field name:* SalesId, *Stored value key:* SalesId, *Input pattern:*.
        - **Element.** *Table name:* DEXCMCustomers, *Field name:* ItemId, *Element name:* ProductNumber, *Optional:* No, *Value:*, *Input pattern:*.
        - **Element.** *Table name:* DEXCMCustomers, *Field name:* Name, *Element name:* ProductName, *Optional:* No, *Value:*, *Input pattern:*.
        - **Element.** *Table name:* DEXCMCustomers, *Field name:* LineAmount, *Element name:* LineAmount, *Optional:* No, *Value:*, *Input pattern:*.
        - **Element.** *Table name:* DEXCMCustomers, *Field name:* LineDisc, *Element name:* LineDiscount, *Optional:* No, *Value:*, *Input pattern:*.
        - **Element.** *Table name:* DEXCMCustomers, *Field name:* QtyOrdered, *Element name:* Quantity, *Optional:* No, *Value:*, *Input pattern:*.
        - **Element.** *Table name:* DEXCMCustomers, *Field name:* ReceiptDateRequested, *Element name:* DeliveryDate, *Optional:* No, *Value:*, *Input pattern:*.
      - **Table end.** *Table name:* DEXCMSalesOrderLine:.
    - **Table end.** *Table name:* DEXCMSalesOrderHeader.

After you have successfully entered all the above values your *XML inbound instructions* tree should look like this:



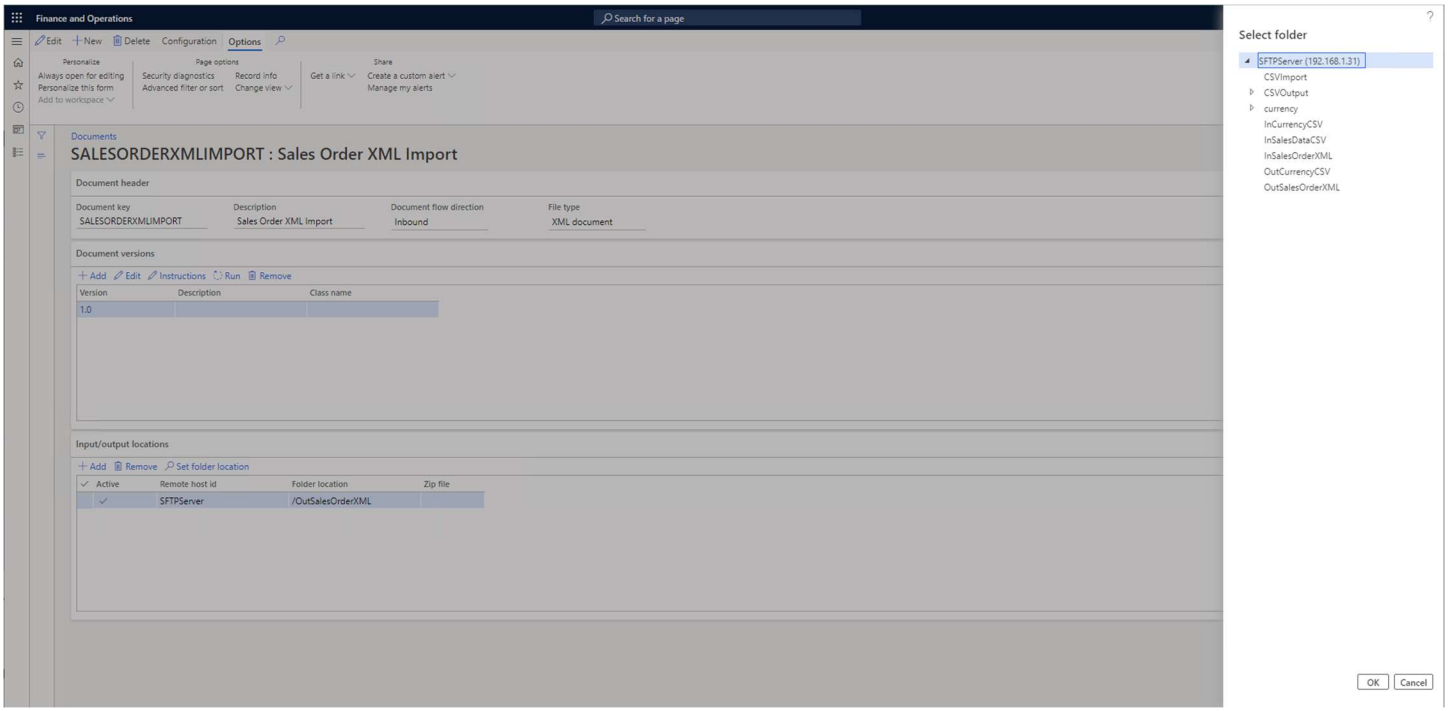
Exit the *XML inbound instructions* form.

In the *Input/output locations* grid you must add a Remote host, set that Remote host's *Folder location* and set that Remote host to *Active*. This will vary according to the setup on your system.



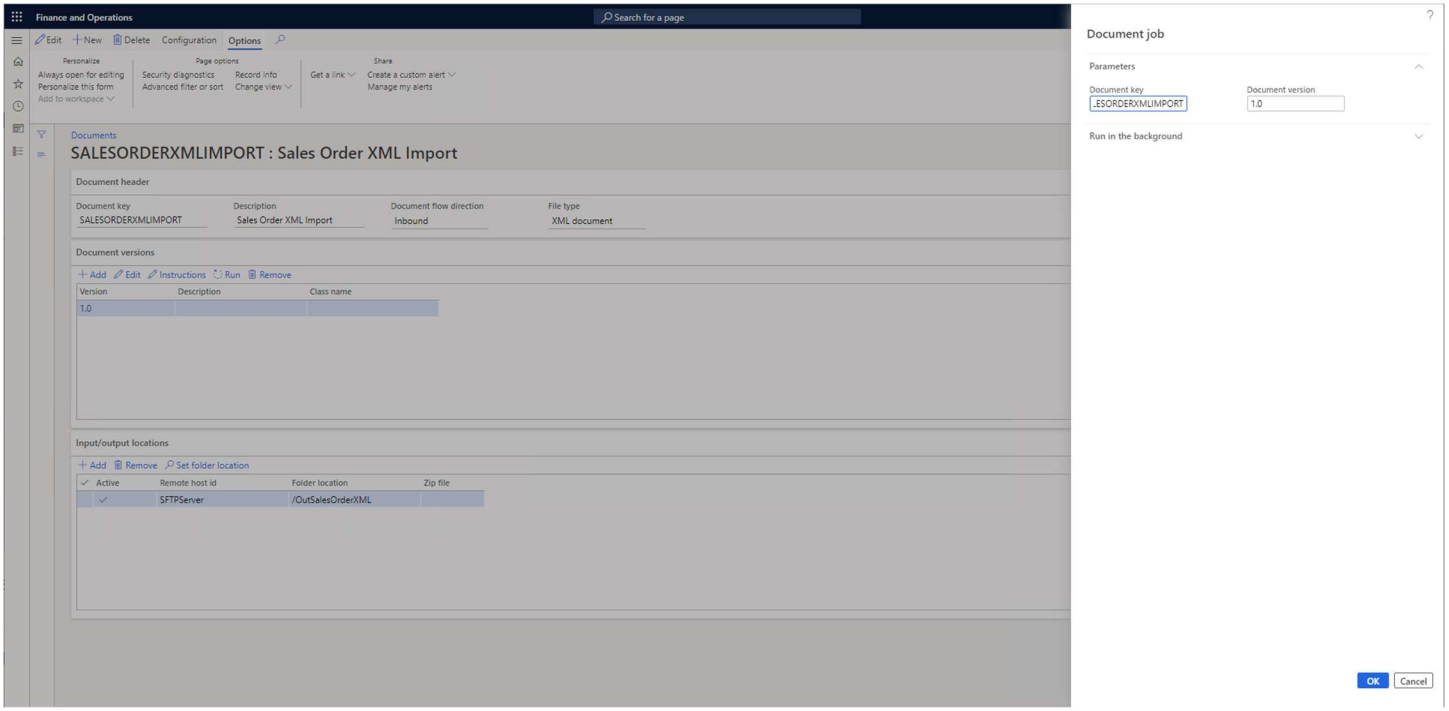
To set the *Folder location* use the *Set folder location* button because the field on the grid is read only.

The *Set folder location* button will bring up a dialog like this one and you will select your destination folder from there:



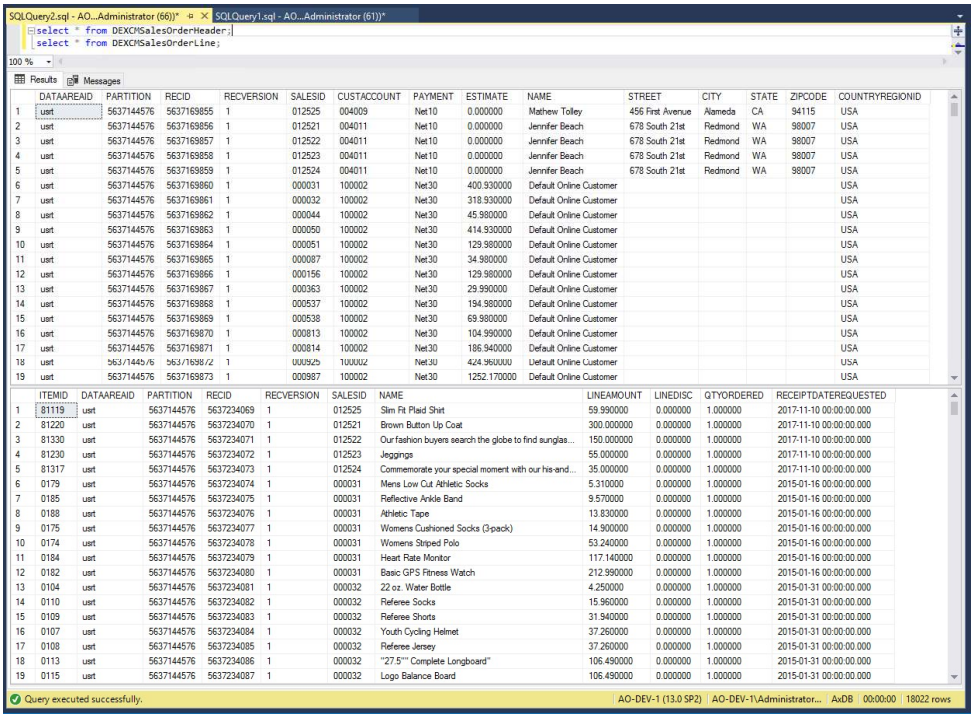
Once you have selected the destination folder click on the *OK* button to exit the dialog.

The Document and Document Version should now be configured correctly. Go to the *Document versions* grid, select version 1.0 and click on the *Run* button to execute version 1.0.



Make sure that *Batch processing* is set to *No*, click on the *OK* button.

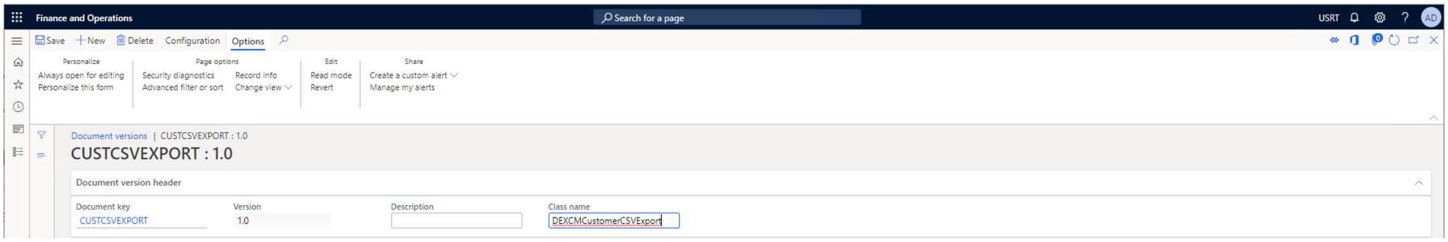
The system will import the XML file into the DEXCMSalesOrderHeader and DEXCMSalesOrderLine D365 tables.



Hard Coded Document Versions

The AODX system allows document integrations that use X++ code instead of Document Version instructions. The downside to this approach is that the Dynamics 365 system must be taken offline to update a Document Version.

With hard coded Document Versions, Document and Document Versions are configured in the exact same way as a Document Version with instructions. But instead of having an instruction tree, the Document Version has the *Class name* property set.



The *Class name* points to a class in an Extension model that references the *Atlantic Oak Document Exchange System*.

Outbound CSV and XML files must create classes that extend the *AODXFileSysOutboundDoc* class. CSV files must override the *writeToCSVFile* method and XML files must override the *writeToXMLFile* method.

Inbound CSV and XML files must create classes that extend the *AODXFileSysInboundDoc* class. CSV files must override the *readFromCSVFile* method and XML files must override the *readFromXMLFile* method.

All four types must also override the *main* method.

Customer Address CSV Export Hard Coded Example

To hard code the instructions for the customer address CSV export you create a class in a model with a reference to the *Atlantic Oak Document Exchange System*. This class must extend the *AODXFileSysOutboundDoc* class and must override the *main* and *writeToCSVFile* methods. The main method is just boiler plate code. The *writeToCSVFile* method uses the *System.IO.StreamWriter* class and its methods to create the output CSV file. The writeToCSVFile method must return a value indicating the number of records that have been written to the file.

Code listing:

```
// <summary>
/// Exports the CustTable table to a CSV
/// file
/// </summary>
class DEXCMCustomerCSVExport extends AODXFileSysOutboundDoc
{
    /// <summary>
    /// Initializes the class and starts the
    /// export
    /// </summary>
    /// <param name="_args">The args class with a parmObject of type AODXArgs</param>
    public static void main(Args _args)
    {
        AODXArgs args = _args.parmObject() as AODXArgs;
        DEXCMCustomerCSVExport dexcmCustomerCSVExport = new
DEXCMCustomerCSVExport(args.paramDocumentKey(), args.paramDocumentVersion());
        dexcmCustomerCSVExport.runOutbound();
    }

    /// <summary>
    /// Allows custom manipulation of file contents
    /// </summary>
    /// <param name="_writer">The writer for the stream</param>
    /// <returns>
    /// The number of rows written
    /// </returns>
    public int writeToCSVFile(System.IO.StreamWriter _writer, AODXCSVOutboundConfiguration
_AODXCSVOutboundConfiguration)
    {
        CustTable custTable;
        DirPartyTable dirPartyTable;
        LogisticsLocation logisticsLocation;
        LogisticsPostalAddress logisticsPostalAddress;
        int recCount = 0;
        str fieldMarker = _AODXCSVOutboundConfiguration.FieldMarker;
        str separator = this.getSeparator(_AODXCSVOutboundConfiguration);

        _writer.writeLine('Generated with X++');
        _writer.WriteLine(
                fieldMarker + 'Account Number' + fieldMarker + separator +
                fieldMarker + 'Name' + fieldMarker + separator +
                fieldMarker + 'Address' + fieldMarker + separator +
                fieldMarker + 'City' + fieldMarker + separator +
                fieldMarker + 'State' + fieldMarker + separator +
                fieldMarker + 'Postal Code' + fieldMarker + separator +
                fieldMarker + 'Country' + fieldMarker
                );

        while select forupdate AccountNum, DEXCMExported, Party from custTable
            join Name, PrimaryAddressLocation from dirPartyTable
            join RecId from logisticsLocation
```

```
            join Street, City, State, ZipCode, CountryRegionId, Location, ValidTo from
logisticsPostalAddress
            where
                custTable.DEXCMExported == NoYes::No &&
                custTable.Party == dirPartyTable.RecId &&
                logisticsLocation.RecId == dirPartyTable.PrimaryAddressLocation &&
                logisticsPostalAddress.Location == logisticsLocation.RecId &&
                logisticsPostalAddress.ValidTo == 2154-12-31T23:59:59
        {

            _writer.WriteLine(
                fieldMarker + custTable.AccountNum + fieldMarker + separator +
                fieldMarker + dirPartyTable.Name + fieldMarker + separator +
                fieldMarker + logisticsPostalAddress.Street + fieldMarker + separator +
                fieldMarker + logisticsPostalAddress.City + fieldMarker + separator +
                fieldMarker + logisticsPostalAddress.State + fieldMarker + separator +
                fieldMarker + logisticsPostalAddress.ZipCode + fieldMarker + separator +
                fieldMarker + logisticsPostalAddress.CountryRegionId + fieldMarker
                );

            ttsbegin;
            custTable.DEXCMExported = NoYes::Yes;
            custTable.update();
            ttscommit;
            recCount++;
        }

        return recCount;
    }

    /// <summary>
    /// Gets the field separator
    /// </summary>
    /// <param name="_AODXXMLOutboundConfiguration">An AODXXMLOutboundConfiguration buffer</param>
    /// <returns>
    /// The field separator
    /// </returns>
    public str getSeparator(AODXCSVOutboundConfiguration _AODXCSVOutboundConfiguration)
    {
        if (_AODXCSVOutboundConfiguration.SeparatorType == AODXSeparatorType::Character)
        {
            return _AODXCSVOutboundConfiguration.Separator;
        }
        else
        {
            return num2Char(_AODXCSVOutboundConfiguration.SeparatorCode);
        }
    }

}
```

Customer Address CSV Import Hard Coded Example

To hard code the instructions for the customer address CSV import you create a class in a model with a reference to the *Atlantic Oak Document Exchange System*. This class must extend the *AODXFileSysInboundDoc* class and must override the *main* and *readFromCSVFile* methods. The main method is just boiler plate code. The *readFromCSVFile* method uses the *System.IO.StreamReader* class and its methods to read from the CSV file.

Code listing:

```
/// <summary>
/// Imports a Customer information CSV file
/// file
/// </summary>
class DEXCMCustomerCSVImport extends AODXFileSysInboundDoc
{
    /// <summary>
    /// Initializes the class and starts the
    /// import
    /// </summary>
    /// <param name="_args">The args class with a parmObject of type AODXArgs</param>
    public static void main(Args _args)
    {
        AODXArgs args = _args.parmObject() as AODXArgs;
        DEXCMCustomerCSVImport dexcmCustomerCSVImport = new
DEXCMCustomerCSVImport(args.paramDocumentKey(), args.paramDocumentVersion());
        dexcmCustomerCSVImport.runInbound();
    }

    /// <summary>
    /// Reads from CSV using custom code in an extension class,
    /// must be overridden in the extension class
    /// </summary>
    /// <param name="_reader">The stream reader</param>
    /// <param name="_AODXCSVInboundConfiguration">An AODXCSVInboundConfiguration buffer</param>
    public void readFromCSVFile(System.IO.StreamReader _reader, AODXCSVInboundConfiguration
_AODXCSVInboundConfiguration)
    {
        int i;
        System.String line;
        System.String[] lineArr;
        for (i = 1; i <= _AODXCSVInboundConfiguration.SkipLines; i++)
        {
            _reader.ReadLine();
        }
        line = _reader.ReadLine();
        RecordInsertList recordInsertList = new RecordInsertList(tableNum(DEXCMCustomers));
        while (line != null)
        {
            lineArr = line.Split(this.getSeparator(_AODXCSVInboundConfiguration));
            if (strLen(_AODXCSVInboundConfiguration.FieldMarker) == 1)
            {
                System.String tmpString;
```

```
                for (i = 0; i <= lineArr.Length - 1; i++)
                {
                    tmpString = lineArr.GetValue(i);
                    if (tmpString.StartsWith(_AODXCSVInboundConfiguration.FieldMarker))
                    {
                        tmpString = tmpString.Substring(1, tmpString.Length - 1);
                    }
                    if (tmpString.EndsWith(_AODXCSVInboundConfiguration.FieldMarker))
                    {
                        tmpString = tmpString.Substring(0, tmpString.Length - 1);
                    }
                    lineArr.SetValue(tmpString, i);
                }
            }
            DEXCMCustomers dexcmCustomers;
            dexcmCustomers.clear();
            dexcmCustomers.AccountNum = lineArr.GetValue(0);
            dexcmCustomers.Name = lineArr.GetValue(1);
            dexcmCustomers.Street = lineArr.GetValue(2);
            dexcmCustomers.City = lineArr.GetValue(3);
            dexcmCustomers.State = lineArr.GetValue(4);
            dexcmCustomers.ZipCode = lineArr.GetValue(5);
            dexcmCustomers.CountryRegionId = lineArr.GetValue(6);
            recordInsertList.add(dexcmCustomers);
            line = _reader.ReadLine();
        }
        recordInsertList.insertDatabase();
    }

    /// <summary>
    /// Gets the field separator
    /// </summary>
    /// <param name="_AODXXMLOutboundConfiguration">An AODXXMLOutboundConfiguration buffer</param>
    /// <returns>
    /// The field separator
    /// </returns>
    public str getSeparator(AODXCSVInboundConfiguration _AODXCSVInboundConfiguration)
    {
        if (_AODXCSVInboundConfiguration.SeparatorType == AODXSeparatorType::Character)
        {
            return _AODXCSVInboundConfiguration.Separator;
        }
        else
        {
            return num2Char(_AODXCSVInboundConfiguration.SeparatorCode);
        }
    }

}
```

Sales Order XML Export Hard Coded Example

To hard code the instructions for the sales order XML export you create a class in a model with a reference to the *Atlantic Oak Document Exchange System*. This class must extend the *AODXFileSysOutboundDoc* class and must override the *main* and *writeToXMLFile* methods. The main method is just boiler plate code. The *writeToXMLFile* method uses the *System.Xml.XmlWriter* class and its methods to create the output XM file. The *writeToXMLFile* method must return a value indicating the number of records that have been written to the file.

Code listing:

```
/// <summary>
/// Exports sales orders to an XML file
/// </summary>
class DEXCMSalesOrderXMLExport extends AODXFileSysOutboundDoc
{
    /// <summary>
    /// Initializes the class and starts the
    /// export
    /// </summary>
    /// <param name="_args">The args class with a parmObject of type AODXArgs</param>
    public static void main(Args _args)
    {
        AODXArgs args = _args.parmObject() as AODXArgs;
        DEXCMSalesOrderXMLExport dexcmSalesOrderXMLExport = new
DEXCMSalesOrderXMLExport(args.paramDocumentKey(), args.paramDocumentVersion());
        dexcmSalesOrderXMLExport.runOutbound();
    }

    /// <summary>
    /// Allows custom manipulation of file contents
    /// </summary>
    /// <param name="_writer">The XML writer</param>
    /// <param name="_AODXXMLOutboundConfiguration">An AODXXMLOutboundConfiguration buffer</param>
    /// <returns>
    /// The number of records written
    /// </returns>
    public int writeToXMLFile(System.Xml.XmlWriter _writer, AODXXMLOutboundConfiguration
_AODXXMLOutboundConfiguration)
    {
        SalesTable salesTable;
        CustTable custTable;
        DirPartyTable dirPartyTable;
        LogisticsPostalAddress logisticsPostalAddress;
        System.DateTime soDate;
        int lineCount = 0;

        _writer.WriteStartElement('SalesOrders');

        while select forupdate SalesId, CustAccount, Payment, Estimate, CreatedDateTime, DEXCMExported
from salesTable
            join AccountNum, Party from custTable
            join Name, PrimaryAddressLocation from dirPartyTable
```

```
            join Street, City, State, ZipCode, CountryRegionId, Location, ValidTo from
logisticsPostalAddress
            where
                salesTable.DEXCMExported == NoYes::No &&
                SalesTable.CustAccount == CustTable.AccountNum &&
                custTable.Party == dirPartyTable.RecId &&
                logisticsPostalAddress.Location == dirPartyTable.PrimaryAddressLocation &&
                logisticsPostalAddress.ValidTo == 2154-12-31T23:59:59
        {
            _writer.WriteStartElement('SalesOrder');
            _writer.WriteAttributeString('OrderNumber', salesTable.SalesId);
            soDate = Global::utcDateTime2SystemDateTime(salesTable.CreatedDateTime);
            _writer.WriteElementString('Date', soDate.ToString());
            _writer.WriteElementString('PaymentTerms', salesTable.Payment);
            _writer.WriteElementString('TotalAmount', System.Convert::ToString(salesTable.Estimate));

            _writer.WriteStartElement('CustomerInfo');
            _writer.WriteAttributeString('CustomerAccount', salesTable.CustAccount);
            _writer.WriteElementString('CustomerName', dirPartyTable.Name);
            _writer.WriteElementString('Address', logisticsPostalAddress.Street);
            _writer.WriteElementString('City', logisticsPostalAddress.City);
            _writer.WriteElementString('State', logisticsPostalAddress.State);
            _writer.WriteElementString('PostalCode', logisticsPostalAddress.ZipCode);
            _writer.WriteElementString('Country', logisticsPostalAddress.CountryRegionId);
            _writer.WriteEndElement(); //CustomerInfo

            _writer.WriteStartElement('Lines');
            this.writeLines(SalesTable.SalesId, _writer);
            _writer.WriteEndElement(); //Lines
            _writer.WriteEndElement(); //SalesOrder

            ttsbegin;
            salesTable.DEXCMExported = NoYes::Yes;
            salesTable.update();
            ttscommit;

            lineCount++;

        }

        _writer.WriteEndElement(); //SalesOrders

        return lineCount;
    }

    /// <summary>
    /// Writes the sales order lines
    /// </summary>
    /// <param name="_salesId">The sales order number</param>
    /// <param name="_writer">The XML writer</param>
    public void writeLines(str _salesId, System.Xml.XmlWriter _writer)
    {
        SalesLine salesLine;
        System.DateTime slDeliveryDate;
        while select ItemId, Name, LineAmount, LineDisc, QtyOrdered, ReceiptDateRequested from
salesLine where
```

```
                salesLine.SalesId == _salesId
        {
            _writer.WriteStartElement('Line');
            _writer.WriteElementString('ProductNumber', salesLine.ItemId);
            _writer.WriteElementString('ProductName', salesLine.Name);
            _writer.WriteElementString('LineAmount', System.Convert::ToString(salesLine.LineAmount));
            _writer.WriteElementString('LineDiscount', System.Convert::ToString(salesLine.LineDisc));
            _writer.WriteElementString('Quantity', System.Convert::ToString(salesLine.QtyOrdered));
            slDeliveryDate = salesLine.ReceiptDateRequested;
            _writer.WriteElementString('DeliveryDate', slDeliveryDate.ToString());
            _writer.WriteEndElement();
        }
    }

}
```

Sales Order XML Import Hard Coded Example

To hard code the instructions for the sales order XML import you create a class in a model with a reference to the *Atlantic Oak Document Exchange System*. This class must extend the *AODXFileSysInboundDoc* class and must override the *main* and *readFromXMLFile* methods. The main method is just boiler plate code. The *readFromXMLFile* method uses the *System.Xml.XmlDocument* class and its methods to read from the XML file.

Code listing:

```
/// <summary>
/// Imports Sales orders in XML files
/// file
/// </summary>
class DEXCMSalesOrderXMLImport extends AODXFileSysInboundDoc
{
    /// <summary>
    /// Initializes the class and starts the
    /// import
    /// </summary>
    /// <param name="_args">The args class with a parmObject of type AODXArgs</param>
    public static void main(Args _args)
    {
        AODXArgs args = _args.parmObject() as AODXArgs;
        DEXCMSalesOrderXMLImport dexcmSalesOrderXMLImport = new
DEXCMSalesOrderXMLImport(args.paramDocumentKey(), args.paramDocumentVersion());
        dexcmSalesOrderXMLImport.runInbound();
    }

    /// <summary>
    /// Reads from XML using custom code in an extension class,
    /// must be overridden in the extension class
    /// </summary>
    /// <param name="_document">The xml document</param>
    /// <param name="_AODXXMLInboundConfiguration">An AODXXMLInboundConfiguration buffer</param>
    public void readFromXMLFile(System.Xml.XmlDocument _document, AODXXMLInboundConfiguration
_AODXXMLInboundConfiguration)
    {
        System.Xml.XmlNode salesOrdersNode = _document.GetElementsByTagName('SalesOrders').Item(0);
        int salesOrdersIterator;
        DEXCMSalesOrderHeader dexcmSalesOrderHeader;
        DEXCMSalesOrderLine dexcmSalesOrderLine;

        RecordInsertList headerInsertList = new RecordInsertList(tableNum(DEXCMSalesOrderHeader));
        RecordInsertList lineInsertList = new RecordInsertList(tableNum(DEXCMSalesOrderLine));

        for (salesOrdersIterator = 0; salesOrdersIterator < salesOrdersNode.ChildNodes.Count;
salesOrdersIterator++)
        {
            System.Xml.XmlNode salesOrderNode = salesOrdersNode.ChildNodes.Item(salesOrdersIterator);
            System.Xml.XmlNode customerInfoNode = this.getNode(salesOrderNode, 'CustomerInfo');
            System.Xml.XmlNode salesLinesNode = this.getNode(salesOrderNode, 'Lines');
            dexcmSalesOrderHeader.clear();
```

```
            dexcmSalesOrderHeader.SalesId = this.getNodeAttrStr(salesOrderNode, 'OrderNumber');
            dexcmSalesOrderHeader.Payment = this.getNodeStr(salesOrderNode, 'PaymentTerms');
            dexcmSalesOrderHeader.Estimate = System.Convert::ToDouble(this.getNodeStr(salesOrderNode,
'TotalAmount'));
            dexcmSalesOrderHeader.CustAccount = this.getNodeAttrStr(customerInfoNode,
'CustomerAccount');
            dexcmSalesOrderHeader.Name = this.getNodeStr(customerInfoNode, 'CustomerName');
            dexcmSalesOrderHeader.Street = this.getNodeStr(customerInfoNode, 'Address');
            dexcmSalesOrderHeader.City = this.getNodeStr(customerInfoNode, 'City');
            dexcmSalesOrderHeader.State = this.getNodeStr(customerInfoNode, 'State');
            dexcmSalesOrderHeader.ZipCode = this.getNodeStr(customerInfoNode, 'PostalCode');
            dexcmSalesOrderHeader.CountryRegionId = this.getNodeStr(customerInfoNode, 'Country');

            int salesOrderLinesIterator;
            for (salesOrderLinesIterator = 0; salesOrderLinesIterator <
salesLinesNode.ChildNodes.Count; salesOrderLinesIterator++)
            {
                System.Xml.XmlNode salesOrderLineNode =
salesLinesNode.ChildNodes.Item(salesOrderLinesIterator);
                dexcmSalesOrderLine.clear();
                dexcmSalesOrderLine.SalesId = this.getNodeAttrStr(salesOrderNode, 'OrderNumber');
                dexcmSalesOrderLine.ItemId = this.getNodeStr(salesOrderLineNode, 'ProductNumber');
                dexcmSalesOrderLine.Name = this.getNodeStr(salesOrderLineNode, 'ProductName');
                dexcmSalesOrderLine.LineAmount =
System.Convert::ToDouble(this.getNodeStr(salesOrderLineNode, 'LineAmount'));
                dexcmSalesOrderLine.LineDisc =
System.Convert::ToDouble(this.getNodeStr(salesOrderLineNode, 'LineDiscount'));
                dexcmSalesOrderLine.QtyOrdered =
System.Convert::ToDouble(this.getNodeStr(salesOrderLineNode, 'Quantity'));
                date receiptDate = System.DateTime::Parse(this.getNodeStr(salesOrderLineNode,
'DeliveryDate'));
                dexcmSalesOrderLine.ReceiptDateRequested = receiptDate;
                lineInsertList.add(dexcmSalesOrderLine);
            }

            headerInsertList.add(dexcmSalesOrderHeader);
        }
        headerInsertList.insertDatabase();
        lineInsertList.insertDatabase();
    }

    System.Xml.XmlNode getNode(System.Xml.XmlNode _node, str _name)
    {
        int i;
        for (i = 0; i < _node.ChildNodes.Count; i++)
        {
            if (_node.ChildNodes.Item(i).Name == _name)
            {
                return _node.ChildNodes.Item(i);
            }
        }
        return null;
    }

    str getNodeStr(System.Xml.XmlNode _node, str _name)
    {
```

```
        int i;
        for (i = 0; i < _node.ChildNodes.Count; i++)
        {
            if (_node.ChildNodes.Item(i).Name == _name)
            {
                return _node.ChildNodes.Item(i).InnerText;
            }
        }
        return '';
    }

    str getNodeAttrStr(System.Xml.XmlNode _node, str _attrName)
    {
        System.Xml.XmlNode attrib = _node.Attributes.GetNamedItem(_attrName);
        return attrib.InnerText;
    }

}
```